ISTANBUL AYDIN UNIVERSITY

THE FACULTY OF ENGINEERING

HALIT AYDIN CAMPUS

FALL – 2020

COM447

OPEN SOURCE SOFTWARES

SIMPLE FILE SYSTEM

Ant Doğa ŞENTÜRK - B1605.010010

Selim CESUR - B1605.010003

DEPARTMENT : Computer Engineering

LECTURER: Dr. Öğr. Üyesi ADEM ÖZYAVAŞ

Introduction

In this project, we created a simple file system. File system is composed of three main parts, super block, inode table and data block table. We create this file system with the name sfs.bin. This file contains the metadata and the actual data fort he file system. We can use five different commands in the file system. “mkdir” command to create directory, “mkfile” command to create a regular file, “cd” command to change directory, “ls” command to print the list of files in the current directory and finally “lsrec” command to list the files in the whole file system.

Code:

We have 5 different global variables in this program. Two of them are to decide to type of the file we create, one is for the size of a datablock, one is for our Simple File System file and the last one is to hold the inode number of the current directory.

```
6   const int REG_FILE = 0;
7   const int DIRECTORY = 1;
8   const int DATA_BLOCK_SIZE = 512;
9
10  FILE *sfs;
11  int currentDirectory = 0;
```

We have three structures for the three main part of our file system. First one is the super block. Super block holds the information of which indices of the inode table and which data blocks are used by our file system. We want to be able to have at most 32 files in our file system, for that reason, we use integer as an inode bitmap. Furthermore, we want to be able to allocate at most 10 data blocks for every file in the system. That’s why we integer array with 10 elements for data bitmap.

Second structure is for the directories or regular files in the file system. It holds the type information, size information and the numbers of the data blocks that file uses.

The last structure is for directory entries. This is where we give name to the files and see in which file they are in by looking their inode number.

```
22  struct super_block {
23      int inode_bitmap;
24      int data_bitmap[10];
25  };
26
27  struct inode_st {
28      int type;
29      int size;
30      int data_block_indices[10];
31  };
32
33  struct dir_ent {
34    char name [28];
35    unsigned int inode_no;
36  };
```

We also create the super block as a global variable since we will use it in many functions.

```
42  struct super_block sb;
```

When we run the program, first the sfs file is gets created. Then, we initializ the file system, then we run the simpleBash() function which is an infinite loop that reads the commands.

```
44  int main() {
45      sfs = fopen("C:\\Users\\selim\\Desktop\\File System\\sfs.bin", "w+");
46      init_fileSystem();
47
48      simpleBash();
49
50      return 0;
51  }
```

When we initialize the file system, we first make everything in the super block 0, because there is no files in the file system yet.

Then we create the first inode named “root”. Everything in the file system will be inside this root. We set the type of the root directory and we set it’s size 0, since there is nothing inside it yet. Lastly, we allocate the first data block for root.

Now that the root uses the first inode in the inode table and first data block in the data block table, we make the first bit of the relevant bitmaps 1.

```
309 void init_fileSystem() {
310     sb.inode_bitmap = 0;
311
312     int i;
313     for(i =0; i<10; i++)
314         sb.data_bitmap[i] = 0;
315
316     struct inode_st root;
317     root.type = DIRECTORY;
318     root.size = 0;
319
320     for(i=0; i<10; i++)
321         root.data_block_indices[i] = 0;
322
323     sb.inode_bitmap = 1;
324     sb.data_bitmap[0] = 1;
```

Every directory will have two directories inside it (. and ..) as default. We create those two directories an entries. Since they are inside the root and the inode number for the root is 0, we set their inode number values to 0. After creating two entries, we add their size to the root’s size.

Now that we set the values of everything related to super block and the root, we write it to the sfs file. We use fwrite command to write the super block, this operation writes the super block to the beginning of the file. Then we write the root next to it using same command.

In order to write the two entries . and .. we jump to the beginning of the data blocks with fseek command. Since root is using the first data block, we write the entries to the beginning of the data block with fwrite command.

```
326     struct dir_ent dot;
327     strcpy(dot.name,".");
328     dot.inode_no = 0;
329 
330     struct dir_ent dotdot;
331     strcpy(dotdot.name,"..");
332     dotdot.inode_no = 0;
333 
334     root.size = sizeof(struct dir_ent)*2;
335 
336     fwrite(&sb, sizeof(struct super_block), 1, sfs);
337     fwrite(&root, sizeof(struct inode_st), 1, sfs);
338 
339     fseek(sfs, sizeof(struct super_block)+(32*sizeof(struct inode_st))+(root.data_block_indices[9]*DATA_BLOCK_SIZE),SEEK_SET);
340 
341     fwrite(&dot, sizeof(struct dir_ent), 1, sfs);
342     fwrite(&dotdot, sizeof(struct dir_ent), 1, sfs);
343 }
```

After initializing the file system. We run our bash, which is an infinite loop. Inside the loop, we take the command as an input from the user. First command is “mkdir”. If the input from the user is mkdir <directory name>, we compare this directory name with the names of all the files in the current directory. If a file with the same name doesn’t exist, we run the mkdir() function to create a new directory.

```
54  void simpleBash() {
55      printf("Write 'help' to see the possible commands.\n");
56      char command[32];
57      while(1) {
58          printf("> ");
59          scanf("%s", command);
60          if(strcmp(command, "mkdir") == 0) {
61              bool fileExists = false;
62              char fileName[32];
63              scanf("%s", fileName);
64              struct inode_st tempInode = getCurrentInode();
65 
66              int numOfEntries = tempInode.size/sizeof(struct dir_ent);
67              fseek(sfs, sizeof(struct super_block)+32*sizeof(struct inode_st)+(tempInode.data_block_indices[9]*DATA_BLOCK_SIZE), SEEK_SET);
68 
69              struct dir_ent tempEntry;
70              int i;
71              for(i=0; i<numOfEntries; i++){
72                  fread(&tempEntry, sizeof(struct dir_ent), 1, sfs);
73                  if(strcmp(tempEntry.name, fileName) == 0) {
74                      printf("File with the same name already exists!\n");
75                      fileExists = true;
76                      break;
77                  }
78              }
79              if(!fileExists) {
80                  mkdir(fileName);
81              }
82          }
```

In the mkdir() function, we first create a new inode structure called newDir. We set the size of it to 0 initially. Then, we user getInodeNo() and getDataBlockNo() functions to return the first empty slot in the inode table and in the data block table.

```
130 void mkdir(char *fileName) {
131     struct inode_st newDir;
132     newDir.type = DIRECTORY;
133     newDir.size = 0;
134 
135     int inode = getInodeNo(sb);
136     int datablock = getDataBlockNo(sb);
```

How we get the first free slot in the inode table is, we do and operation between the inode_bitmap and 1. This operation returns 0 if the least significant bit of the inode_bitmap is 0. If not, we right shift the inode_bitmap by 1 to change the least significant bit of it. Then compare it again until we get the 0. Getting the data block number works the same way.

```
356 int getInodeNo(struct super_block sb) {
357     int i;
358     for(i=0; i<32; i++) {
359         if((sb.inode_bitmap & 1) == 0) {
360             return i;
361         }
362         else {
363             sb.inode_bitmap = sb.inode_bitmap >> 1;
364         }
365     }
366     printf("There is not enough space in the file system!");
367     exit(0);
368 }
369 
370 int getDataBlockNo(struct super_block sb) {
371     int i;
372     int j;
373     for(i=0; i<10; i++) {
374         for(j=0; j<32; j++) {
375             if((sb.data_bitmap[i] & 1) == 0) {
376                 return (i*32)+j;
377             }
378             else {
379                 sb.data_bitmap[i] = sb.data_bitmap[i] >> 1;
380             }
381         }
382     }
383     printf("There is not enough space in the file system!");
384     exit(0);
385 }
```

After we get the inode nuöber and data block number, we set the data block indices for the new directory, and create the (.) and (..) entries same with the root. Since we add the directory entries to new directory, we increase the size of the accordingly.

```
    int i;
    for(i=0; i<10; i++)
        newDir.data_block_indices[i] = datablock;

    struct dir_ent dot;
    strcpy(dot.name,".");
    dot.inode_no = inode;

    struct dir_ent dotdot;
    strcpy(dotdot.name,"..");
    dotdot.inode_no = currentDirectory;

    newDir.size = sizeof(struct dir_ent)*2;
```

Now that we use the inode and data block that we get for our new directory, we set them to 1 in the relative bitmaps. After updating the bitmaps, we update the super block by writing it to file again.

Another thing we need to update is the size of the current directory because we will add the new directory as a directory entry in the current directory. We add it's size another size of a director entry and then re-write it to the file.

```
    int mask = 1 << inode;
    sb.inode_bitmap = (sb.inode_bitmap & ~mask) | ((1 << inode) & mask);

    int index = datablock/32;
    int p = datablock%32;
    mask = 1 << p;
    sb.data_bitmap[index] = (sb.data_bitmap[index] & ~mask) | ((1 << p) & mask);

    fseek(sfs, 0, SEEK_SET);
    fwrite(&sb, sizeof(struct super_block), 1, sfs);

    struct inode_st tempInode = getCurrentInode();

    tempInode.size += sizeof(struct dir_ent);

    fseek(sfs, sizeof(struct super_block)+(currentDirectory*sizeof(struct inode_st)), SEEK_SET);
    fwrite(&tempInode, sizeof(struct inode_st), 1, sfs);
```

Now, we jump to the correct place in the inode table and write the new directory there. Then, we jump to the data block table and write the two entries (.) and (..).

Lastly, we create a directory entry for the new directory and add it to the data block of the current directory.

```
    fseek(sfs, sizeof(struct super_block)+inode*sizeof(struct inode_st), SEEK_SET);
    fwrite(&newDir, sizeof(struct inode_st), 1, sfs);

    fseek(sfs, sizeof(struct super_block)+(32*sizeof(struct inode_st))+(newDir.data_block_indices[9]*DATA_BLOCK_SIZE), SEEK_SET);
    fwrite(&dot, sizeof(struct dir_ent), 1, sfs);
    fwrite(&dotdot, sizeof(struct dir_ent), 1, sfs);

    struct dir_ent newDirEnt;
    strcpy(newDirEnt.name,fileName);
    newDirEnt.inode_no = inode;

    fseek(sfs, sizeof(struct super_block)+(32*sizeof(struct inode_st))+(tempInode.data_block_indices[9]*DATA_BLOCK_SIZE)+
    (tempInode.size-32), SEEK_SET);

    fwrite(&newDirEnt, sizeof(struct dir_ent), 1, sfs);
}
```

Second command is “mkfile”. This command works exactly the same way with mkdir command. Only difference is that we don’t add (.) and (..) entries to it’s data block. We instead add a text to it. Then, we set the size of it to the size of the text.

```
186 void mkfile(char *fileName) {
187
188     char text[] = "My name is Selim and his name is Doğa. We have met at English preparation school 4 years ago...";
189
190     struct inode_st newFile;
191     newFile.type = REG_FILE;
192     newFile.size = sizeof(text);
```

Third command is “cd”. We use this command the change the current Directory. First, we get the current Directory and see how many files inside it by dividing it’s size by the size of an entry.

```
233 void cd() {
234     char fileName[32];
235     scanf("%s", fileName);
236     bool fileExists = false;
237
238     struct inode_st tempInode = getCurrentInode();
239
240     int numOfEntries = tempInode.size/sizeof(struct dir_ent);
241     fseek(sfs, sizeof(struct super_block)+32*sizeof(struct inode_st)+(tempInode.data_block_indices[9]*DATA_BLOCK_SIZE), SEEK_SET);
```

Then, we iterate through all files and see the directory we want to go does exist. If we find the file, we check it’s type. If it is a directory, we change the global variable current Directory to it’s inode number. If it’s not a directory, we just send a informative message and don’t do anything else. If we don’t find the file, we again print a messaje to inform the user that the file doesn’t exists.

```
243     struct dir_ent tempEntry;
244     int i;
245     for(i=0; i<numOfEntries; i++){
246         fread(&tempEntry, sizeof(struct dir_ent), 1, sfs);
247         if(strcmp(tempEntry.name, fileName) == 0) {
248
249             fileExists = true;
250             fseek(sfs, sizeof(struct super_block)+(tempEntry.inode_no*sizeof(struct inode_st)), SEEK_SET);
251
252             struct inode_st tempInode;
253             fread(&tempInode, sizeof(struct inode_st), 1, sfs);
254
255             if(tempInode.type == DIRECTORY) {
256                 currentDirectory = tempEntry.inode_no;
257             }
258             else {
259                 printf("This file is not a directory!\n");
260             }
261             break;
262         }
263     }
264     if(!fileExists) {
265         printf("No such directory!\n");
266     }
```

Forth command is “ls”. We use ls to print the list of the files in the current Directory. We get the current directory and iterate all the files inside of it’s data block the same way we did in the cd command. This time, while we iterate them, we print them to the console.

```
void ls() {
    struct inode_st tempInode = getCurrentInode();

    int numOfEntries = tempInode.size/sizeof(struct dir_ent);
    fseek(sfs, sizeof(struct super_block)+32*sizeof(struct inode_st)+(tempInode.data_block_indices[9]*DATA_BLOCK_SIZE), SEEK_SET);

    struct dir_ent tempEntry;
    int i;
    for(i=0; i<numOfEntries; i++){
        fread(&tempEntry, sizeof(struct dir_ent), 1, sfs);
        printf("Directory %d: %s\n",i+1, tempEntry.name);
    }
}
```

The last command is “lsrec”. This command -differently from the “ls”- prints the whole file system when we write it. Same with the ls command, we get the current Directory and iterate the files in it’s data block. Instead of printing the file names, we store them in an array to print it later. After storing them, we iterate them again but this time we check their name and see if it’s a directory that we need go inside (directories that are not (.) and (..)). If they are, we change the local current directory variable and run the lsrec function with as this current directory variable is one of it’s arguments. This way lsrec becomes a recursive function that prints the whole file system.

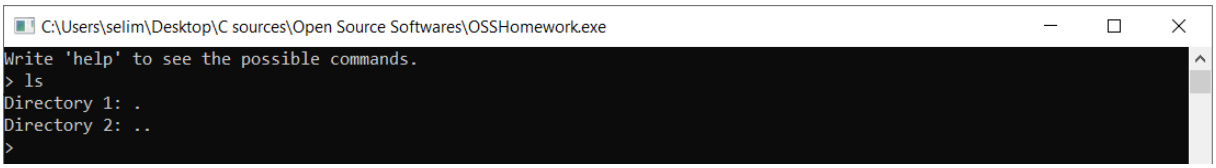
```
void lsrec(int tab, int currentDirectory) {
    struct inode_st tempInode = getCurrentInode();

    if(tempInode.type == DIRECTORY) {
        int numOfEntries = tempInode.size/sizeof(struct dir_ent);
        fseek(sfs, sizeof(struct super_block)+32*sizeof(struct inode_st)+(tempInode.data_block_indices[9]*DATA_BLOCK_SIZE), SEEK_SET);

        int i;
        struct dir_ent entries[numOfEntries];
        for(i=0; i<numOfEntries; i++){
            fread(&entries[i], sizeof(struct dir_ent), 1, sfs);
        }

        int j;
        for(i=0; i<numOfEntries; i++){
            for(j=0; j<tab; j++) {
                printf("\t");
                printf("    ");
            }
            printf("Directory %d: %s\n",i+1, entries[i].name);
            if(strcmp(entries[i].name, ".") != 0 && strcmp(entries[i].name, "..") != 0) {
                currentDirectory = entries[i].inode_no;
                tab++;
                lsrec(tab, currentDirectory);
                tab--;
            }
        }
    }
}
```

This is the function that we used all the commands above to get current directory’s inode:

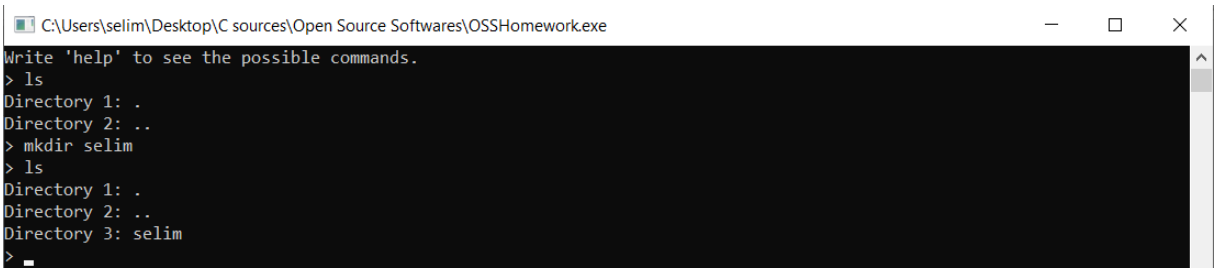
```
struct inode_st getCurrentInode() {
    fseek(sfs, sizeof(struct super_block)+(currentDirectory*sizeof(struct inode_st)), SEEK_SET);
    struct inode_st tempInode;
    fread(&tempInode, sizeof(struct inode_st), 1, sfs);

    return tempInode;
}
```

Sample Run:

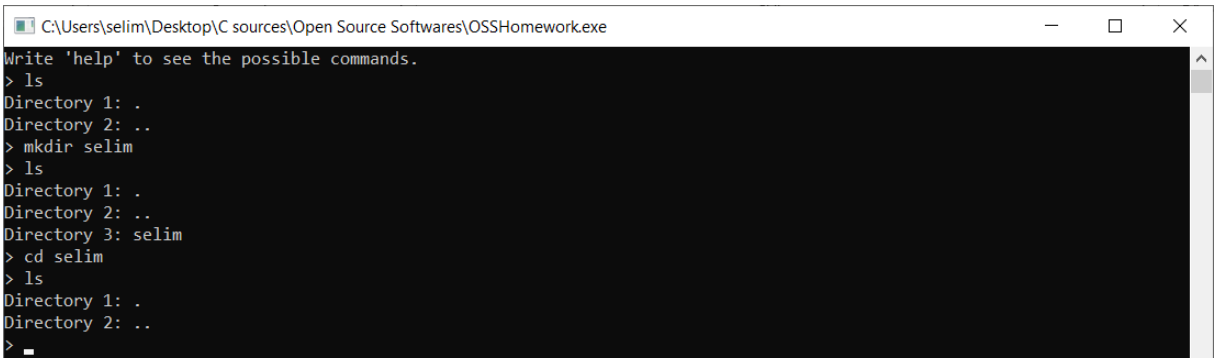
First, we write “ls” to see that there only two directories (.) and (..) in the root.



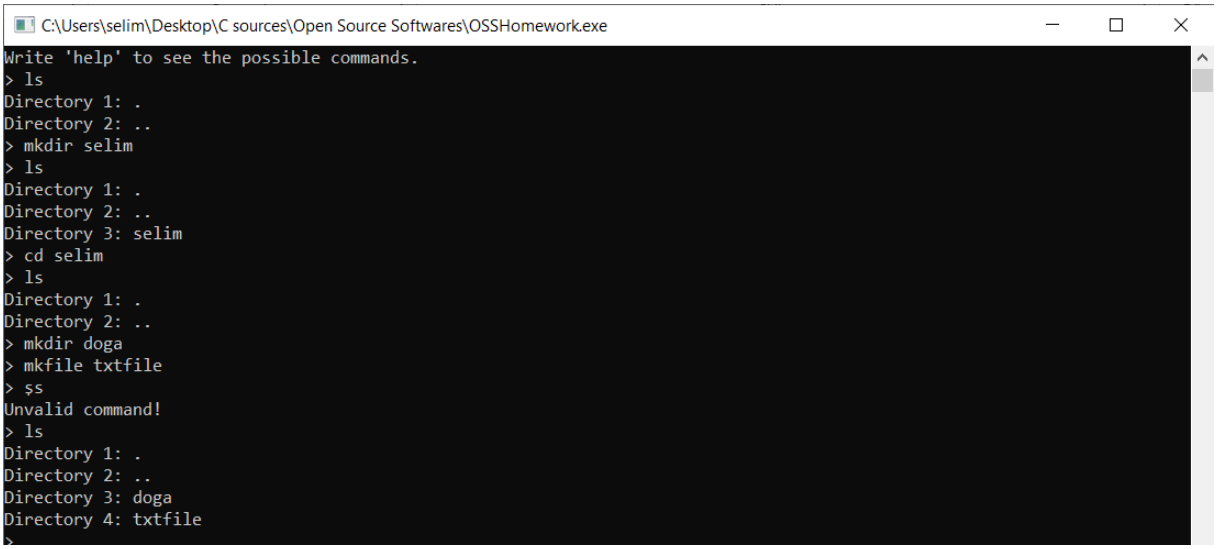
Then, we run “mkdir” command to create a directory named “selim”. If we check with “ls” command, we see that it is created correctly.



Then, we change the directory to “Selim” with cd command and print what’s inside it with again “ls”.



After that, we create another directory and a regular file inside “Selim”. It says “Unvalid command” if we miss spell it (Oops!).



Lastly, we write “lsrec” to print the whole file system.