**Department of Computer Engineering**

BLG413E – SYSTEM PROGRAMMING Project-3 Report

Group Number : G37

Group Members :

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Surname** |
| 150160537 | Okan | YILDIRIM |
| 150160530 | Özgür | AKTAŞ |
|  |  |  |
|  |  |  |

Instructor : Lec. H. Turgut UYAR

1. **Introduction**

Main aim of the this project is implementing file system by using FUSE in Linux Operating System. With this project we have a chance to develop file system in Userspace by using the information we learned in class and compare cons and pros of developing in user-space with the previous projects. We experience that studying on user-space is least risky and fastest among the compiling kernel space and using kernel modules even though its performance is worst. In addition, it has advantage to use external libraries.

In this project, we implemented a file system which is constructed according to the a CSV file which contains Turkish postal codes data. In this file, there is six column: CODE, NEIGHBIRHOOD, CITY, DISTRICT, LATITUDE and LONGTITUDE. It is requested that there will be two directories in the top level directory: NAMES and CODES. In Names directory, there will be all city subdirectories. Each city directories will show the its districts as subdirectories and finally each district directory will show its neighborhood as text file. This text file contain data in a form of key-value pairs for each column. On the other hand, CODES directory will display every city code which is the two digits of the postal codes. Each city codes subdirectory will show the its neighborhood code as text file. Moreover, this FUSE-based file system will allow to delete and rename operations for only neighborhood files. These modifications will update the CSV file also. In same manner, any changes on the CSV file will be update file system accordingly. Consequently, general description of the project is given above.

1. **Method and Detailed Explanation of Codes**

Before we have been writing the project, we had checked if the FUSE was installed or not on to our Linux Machine. We saw that its 26th version was installed. Then, we implemented the general architecture of the project. Firstly, we defined the FUSE version and included required libraries. This part of code is given below:

#define FUSE\_USE\_VERSION 26

#include <fuse.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

It is very clear that ***fuse.h*** is mandatory for implementing FUSE. Other three libraries required for using some functions such as ***fopen, fprint, strcat, strcpy, strcmp, memset, malloc*** etc. Using these libraries is are advantage of developing on user-space. Then, we used ***fuse\_operation*** structure which contains definition of functions in FUSE. In ***fuse\_operation*** structure, there is a lot of function can be used such as ***mkdir, chmod, open, read, release, rename*** etc. However, these functions are not mandatory to use and we only used functions that is needed with respect to the project. Actually, we wrote a struct for using these functions in way that mapping our functions we wrote to the these functions on the ***fuse\_operation*** structure. These part of code is given below:

static struct fuse\_operations tpc\_operations = {

.getattr = tpc\_getattr,

.readdir = tpc\_readdir,

.read = tpc\_read,

}

Actually, we should use the ***unlink*** function to delete a file and ***rename*** function to change to name of the file, yet we could not finish on time project unfortunately. We could implement only ***getattr***, ***readdir*** and ***read*** functions.

Then, we wrote the main function to which runs our FUSE. It is given Figure 3 below:

int **main**(int argc, char \*argv[]) {

return fuse\_main(argc, argv, &tpc\_operations, NULL);

}

In order to use data which is read from CSV file systematically, it was required to wrote a structure which contains variables such as city, district, neighborhood etc. We wrote ***tpc*** structure that ***tpc*** represents “Turkish postal codes”. This part of code is given below:

static const int linenumber=36307;//36307

struct tpc {

char code[6];

char neighborhood[30];

char city[20];

char district[30];

char latitude[10];

char longtitude[10];

char text[200];

char cityCode[3];

};

struct tpc \*objs;

static const char \*folder\_path = "/NAMES";

static const char \*folder\_path2 = "/CODES";

Structure of ***tpc*** contains all columns in CSV file as variables. In addition we created ***text*** variable which is used for displaying data in a form of key-value pairs for each column when file is opened. Moreover, we created ***cityCode*** to store first two digits of the postal codes which represent the city. Their size are given according to the maximum number of characters of the column values. It is reasonable that type of longitude and latitude can be double, yet we prefer character array because they are not used in a numeric manner in this project.

There are other global variables with ***tpc*** structure above. In order to end some loop and read file, ***linenumber*** variable is used which represents the number of lines in CSV file. On the other hand, to create the CODES and NAMES directories in top of the file system, I used ***folder\_path*** and ***folder\_path2*** named variables.

Then we wrote the reading file part and assigned the each data on line in CSV file to the variables in the ***tcp*** structure. We wrote this part in main function and this part of code is given below:

int **main**(int argc, char \*argv[]) {

/\*Memory allocation for the all objects,

Actuall object number is 36307, yet we allocate a little bit more\*/

objs = malloc(36319 \* sizeof(struct tpc)); //36307

FILE \*infile; //for reading file, create input file

//declarations of some requried varibles for some operations

char line[100];

char \*templine,\*linePtr;

int i = 0; /\* i will correspond to each line and each object which are mapping to each other\*/

//open file which will be read and read each line in a loop

if ((infile = fopen("postal-codes.csv", "r")) != NULL) {

while(fgets (line, 100, infile) != NULL){ /\* 100 corresponds to number of characters in a line\*/

linePtr = line;

linePtr = strsep (&linePtr, "\n"); /\* get line until come across "/n and assign it to linePtr \*/

/\* Seperate line by "\t" and assign each value to related variable of object.

Do it for each line and each object. In this way construct the each object of structure \*/

templine=strsep(&linePtr,"\t");

strcat(objs[i].code,templine);

templine=strsep(&linePtr,"\t");

strcat(objs[i].neighborhood,templine);

templine=strsep(&linePtr,"\t");

strcat(objs[i].city,templine);

templine=strsep(&linePtr,"\t");

strcat(objs[i].district,templine);

templine=strsep(&linePtr,"\t");

strcat(objs[i].latitude,templine);

templine=strsep(&linePtr,"\t");

strcat(objs[i].longtitude,templine);

/\* to creating a text variable for each object via using snprinf function\*/

snprintf(objs[i].text, sizeof(objs[0].text),

"code: %s\nneigborhood: %s\ncity: %s\ndistrict: %s\nlatitude: %s\nlongtitude: %s\n",

objs[i].code, objs[i].neighborhood, objs[i].city, objs[i].district, objs[i].latitude,

objs[i].longtitude);

/\*result of this function in such a form:

code: 34398

neighborhood: Maslak

city: Istanbul

district: Sariyer

latitude: 41.1083

longitude: 29.0183

\*/

/\*get the first two digit of postal code and assign it to the citycode\*/

strncpy(objs[i].cityCode, objs[i].code, 2);

i++;

}

}

else { // if file is not found or not opened

printf("File is not found!!");

}

return fuse\_main(argc, argv, &tpc\_operations, NULL);

}

In this part of code above, we read the CSV file and assign each data related variable of object. To obtain line, ***fgets*** function; to separate the line by new line and column (“\n”and ”\t”) ***strsep*** function; to assign value to object variable, ***strcat*** function; to create temp variable in a determined form, ***snprintf*** function; to copy first two digit of postal code to city code ***strncpy*** function is used. In a loop, file is read line by line and all lines separated one by one so that each data assign to variable of object. Variables are assigned to ***text*** and ***cityCode*** variables after some modifications on the strings. Detailed description of code is given with comment line above. I want to point out that we did reading file operation in a main function. However, it is required that any changes to the CSV file will be instantly visible through the FUSE-based file system and vice versa according to the project description. This is not possible with our method. We should write ***open*** function and reading file operation should be done in this function. In this way, at the each calling of open function reading operation will work and updates are done here instead of reading file once in main function as our wrote. Unfortunately, we noticed proper method after submission of the project.

Then we wrote ***tpc\_getattr*** function which is one the most important function in FUSE. Main purpose of ***getattr*** function is to determine whether the given path which is one of the parameter of function, is a directory or file. In addition , to determine permissions about the file or directories, ***getattr*** function is used. Our ***tpc\_getattr*** function and its detailed explanations with comment lines are given below:

static int **tpc\_getattr**(const char \*path, struct stat \*st) {

//1st parameter is path: path of the file

//second paramerer is stat structure contain file's attributes

/\*clear the content of stat buffer \*/

memset(st, 0, sizeof(struct stat));

int i=0;

/\*Firstly we checked the root directory

If the path is root \*/

if (strcmp(path, "/") == 0) {

/\* st\_mode determine if the path is folder or regular file

and it determine permissions of file

S\_IFDIR macro means path is directory

S\_IFREG macro means pat is regular file

0755 is permission bits and

it means only owner has authoritiy to write whereas

the group ant others can oly read and execute directory

\*/

st->st\_mode = S\_IFDIR | 0755;

/\*st\_link determine the number of hardlinks

2 corresponds to directory itself and parent directory

in other words "./" and "../"

Totally links are these two and other subdirectories\*/

st->st\_nlink = 2;

}

/\*Secondly we checked if the path is "/NAMES" \*/

else if (strcmp(path, folder\_path) == 0) {

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

/\*Thirdly, we checked if the path is "/CODES " \*/

else if (strcmp(path, folder\_path2) == 0) {

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

/\* Until now path names are specified in advance, but now

path names is nor known in advance. It is up to file has been read\*/

else{

while(i<linenumber){ // linenumber=36307

/\* Firstly we create the all possible path

can corresponds to directories and files

in a loop check all objects \*/

char \*districtdir;

districtdir=(char\*)malloc(sizeof(char)\*50); //memory allocation

memset(districtdir,0,sizeof(char)\*50); //clear the content just in case

char \*citydir;

citydir=(char\*)malloc(sizeof(char)\*30);

memset(citydir,0,sizeof(char)\*30);

char \*neighborhooddir;

neighborhooddir=(char\*)malloc(sizeof(char)\*60);

memset(neighborhooddir,0,sizeof(char)\*60);

strcat(citydir,folder\_path);

strcat(citydir,"/");

strcat(citydir,objs[i].city);

/\* EXAMPLE: "/NAMES/Istanbul" \*/

strcat(districtdir,citydir);

strcat(districtdir,"/");

strcat(districtdir,objs[i].district);

/\* EXAMPLE :"/NAMES/Istanbul/Sariyer" \*/

strcat(neighborhooddir,districtdir);

strcat(neighborhooddir,"/");

strcat(neighborhooddir,objs[i].neighborhood);

/\* EXAMPLE: "/NAMES/Istanbul/Sariyer/Maslak" \*/

char \*cityCodedir;

cityCodedir = (char \*) malloc(sizeof(char) \* 10);

memset(cityCodedir, 0, sizeof(char) \* 10);

char \*codedir;

codedir = (char \*) malloc(sizeof(char) \* 18);

memset(codedir, 0, sizeof(char) \* 18);

strcat(cityCodedir, folder\_path2);

strcat(cityCodedir, "/");

strcat(cityCodedir, objs[i].cityCode);

/\* EXAMPLE: "/CODES/34" \*/

strcat(codedir,cityCodedir);

strcat(codedir, "/");

strcat(codedir, objs[i].code);

/\* EXAMPLE: "/CODES/34/34398" \*/

/\* According to the project description,

neighborhood and neighborhood postal codes should be file

city and district and citycode should be directory\*/

if(strcmp(path,neighborhooddir)==0||strcmp(path,codedir)==0){

/\*S\_IFREG means Regular file\*/

/\*S\_IFREG means Regular file\*/

/\*0444 means permission for only reading\*/

st->st\_mode = S\_IFREG | 0444;

// number of link =1 beacuse it is file

st->st\_nlink = 1;

// st\_size is size of the file in terms of byte

st->st\_size = (off\_t)strlen(objs[i].text);

break;

/\* if path is found break the loop\*/

}

else if(strcmp(path,citydir)==0||strcmp(path,districtdir)==0

||strcmp(path,cityCodedir)==0){

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

break;

}

i++;

}

}

return 0;

}

In ***tpc\_getattr*** function get the attributes of requested path, yet our method in ***tpc\_getattr*** function is not good in terms of time complexity because when checking if the path is directory or file, it traverses all objects until found the path and did lots of modifications on strings for each loop. We submit our project as above yet, it can be ***tpc\_getattr*** optimized such below:

static int **tpc\_getattr**(const char \*path, struct stat \*st) {

memset(st, 0, sizeof(struct stat));

int i=0;

int freq=0;

char tempPath[60];

memset(tempPath, 0,60);

strcpy(tempPath,path);

/\*find the frequency of '/' in path \*/

for(i = 0; tempPath[i] != '\0'; ++i)

{

if('/'== tempPath[i])

++freq;

}

char \*parentPath = strtok(tempPath, "/");

if (strcmp(path, "/") == 0) {

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

else if (strcmp(path, folder\_path) == 0) {

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

else if (strcmp(path, folder\_path2) == 0) {

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

/\* EXAMPLE: /NAMES/Istanbul/Sariyer/Maslak \*/

else if(strcmp(parentPath, folder\_path+1) == 0 && freq==4){

st->st\_mode = S\_IFREG | 0444;

st->st\_nlink = 1;

st->st\_size = (off\_t)strlen(objs[i].text);

}

/\* EXAMPLE: /CODES/34/34398 \*/

else if(strcmp(parentPath, folder\_path2+1) == 0 && freq==3){

st->st\_mode = S\_IFREG | 0444;

st->st\_nlink = 1;

st->st\_size = (off\_t)strlen(objs[i].text);

}

else{

st->st\_mode = S\_IFDIR | 0755;

st->st\_nlink = 2;

}

return 0;

}

Instead of checking all object in loop, just count the number of ‘/’ character in path, we can determine if the path is directory or file. Because we knew from the project description, which subdirectories should be file or directories. Only there are two cases for file, otherwise path will be directories and in this way we get rid of the loop. Therefore, performance of the program can get better.

Then, we wrote ***readdir*** function of FUSE. This function reads the directory and create the subdirectories or files for given path. In similar way to ***getattr*** function, we firstly read directories which is determined in advanced such as root (mounted directory), ”NAMES” and “CODES”. Then, we create the all possible path can corresponds to directories and files in a loop by checking all objects. Detailed code description and ***tpc\_readdir*** function is given below:

static int **tpc\_readdir**(const char \*path, void \*buf, fuse\_fill\_dir\_t filler,

off\_t offset, struct fuse\_file\_info \*fi) {

int i = 0;

// check if the path=root

if (strcmp(path, "/") == 0) {

filler(buf, ".", NULL, 0); // Current Directory

filler(buf, "..", NULL, 0); // Parent Directory

filler(buf, folder\_path + 1, NULL, 0);// create "/NAMES" subdirectory"

filler(buf, folder\_path2 + 1, NULL, 0);// create "/CODES" subdirectory"

}

// if path="/NAMES" create all cities

else if (strcmp(path, folder\_path) == 0) {

i = 0;

char tempCity[20]; // is used for checking city name is changed or not

memset(tempCity,0,sizeof(char)\*20);

filler(buf, objs[i].city, NULL, 0); // create first city in /NAMES dir.

strcpy(tempCity, objs[i].city);

//traverse on all object if name of city is changed, create new city

while (i < linenumber) {

if (strcmp(tempCity, objs[i].city) == 0) {

// if city is equals to temp city do nothing

i++;

} else { //otherwise create th city and update the tempcity

filler(buf, objs[i].city, NULL, 0);

memset(tempCity,0,sizeof(char)\*20);

strcpy(tempCity, objs[i].city);

i++;

}

}

}

//if path="/CODES" create all city codes

// Strategy is the same as if path="/NAMES" create all cities above

else if (strcmp(path, folder\_path2) == 0) {

pathIsFound=true;

int i = 0;

char tempCityCode[3];

memset(tempCityCode,0,sizeof(char)\*3);

filler(buf, objs[i].cityCode, NULL, 0); //create first citycode in /CODES dir.

strcpy(tempCityCode, objs[i].cityCode);

while (i < linenumber) { //36327

if (strcmp(tempCityCode, objs[i].cityCode) == 0) {

i++;

} else {

filler(buf, objs[i].cityCode, NULL, 0);

memset(tempCityCode,0,sizeof(char)\*3);

strcpy(tempCityCode, objs[i].cityCode);

i++;

}

// i++;

}

}

/\* if path = city directory as an example: "/NAMES/Istanbul"

create all districts of this city\*/

else {

i=0;

//traverse on all object if name of city is changed, create new city

while (i < linenumber) {

/\* create required varibles and

form all possible directories for each iteration \*/

char tempDistrict[30];

memset(tempDistrict, 0, sizeof(char) \* 30);

strcat(tempDistrict, "");

char \*citydir;

citydir = (char \*) malloc(sizeof(char) \* 30);

memset(citydir, 0, sizeof(char) \* 30);

char tempCity[20];

memset(tempCity, 0, sizeof(char) \* 20);

strcpy(tempCity, objs[i].city);

strcat(citydir, folder\_path);//folder\_path =/NAMES

strcat(citydir, "/");

strcat(citydir, objs[i].city);

/\*EXAMPEL:/NAMES/Istanbul \*/

// if path = city directory as an example: "/NAMES/Istanbul

if (strcmp(path, citydir) == 0) {

// crate all districts of city until city is changed

while (!(strcmp(tempCity, objs[i].city))) {

if (strcmp(tempDistrict, objs[i].district) == 0) {

i++;

// if distrcit are same do nothing

} else {// otherwise create district and update tempDistrict

memset(tempDistrict, 0, sizeof(char) \* 30);

strcpy(tempDistrict, objs[i].district);

filler(buf, objs[i].district, NULL, 0);

i++;

}

}

// if path is found and all districts are created break the loop

break;

}

i++;

}

}

/\* if path = district directory as an example: "/NAMES/Istanbul/Sariyer"

create all neighborhoods of this district\*/

/\* Strategy is similar above if path = city directory as an example: "/NAMES/Istanbul\*/

i=0;

//traverse on all object if name of city is changed, create new city

while (i < linenumber) {

/\* create required varibles and

form all possible directories for each iteration \*/

char tempNeighborhood[40];

memset(tempNeighborhood, 0, sizeof(char) \* 40);

char tempDistrict[40];

memset(tempDistrict, 0, sizeof(char) \* 40);

strcat(tempDistrict, "");

char \*districtdir;

districtdir = (char \*) malloc(sizeof(char) \* 50);

memset(districtdir, 0, sizeof(char) \* 50);

strcpy(tempDistrict, objs[i].district);

strcat(districtdir, folder\_path);

strcat(districtdir, "/");

strcat(districtdir, objs[i].city);

strcat(districtdir, "/");

strcat(districtdir, objs[i].district);

/\*EXAMPEL:/NAMES/Istanbul/Sariyer \*/

// if path = city directory as an example: "/NAMES/Istanbul/Sariyer

if (strcmp(path, districtdir) == 0) {

// crate all neighborhood of district until distrcict is changed

while (!(strcmp(tempDistrict, objs[i].district))) {

if (strcmp(tempNeighborhood, objs[i].neighborhood) == 0) {

i++; // if neighborhood is the same do nothing

} else {// otherwise create neigborhood and update tempNeighborhood

memset(districtdir, 0, sizeof(char) \* 40);

strcpy(tempNeighborhood, objs[i].neighborhood);

filler(buf, objs[i].neighborhood, NULL, 0);

i++;

}

}

// if path is found and all neighborhood are created break the loop

break;

}

i++;

}

// }

i = 0;

/\* if path = citycode create all postal codes for this city,

as an example if path="CODES/34" create all postal codes for 34\*/

/\* Strategy is the same as creating district for

if path = city directory as an example: "/NAMES/Istanbul\*/

while (i < linenumber) {

char tempCode[20] = "";

char \*cityCodedir;

cityCodedir = (char \*) malloc(sizeof(char) \* 20);

memset(cityCodedir, 0, sizeof(char) \* 20);

char tempCityCode[20];

strcpy(tempCityCode, objs[i].cityCode);

//

strcat(cityCodedir, folder\_path2);

strcat(cityCodedir, "/");

strcat(cityCodedir, objs[i].cityCode);

if (strcmp(path, cityCodedir) == 0) {

while (!(strcmp(tempCityCode, objs[i].cityCode))) {

if (strcmp(tempCode, objs[i].code) == 0) {

i++;

} else {

strcpy(tempCode, objs[i].code);

filler(buf, objs[i].code, NULL, 0);

i++;

}

}

strcpy(tempCityCode, objs[i].cityCode);

break;

}

i++;

}

// }

return 0;

}

Finally we wrote the read function for reading files. According to project description, each district directory will show its neighborhood as text file and each city codes subdirectory will show the its neighborhood code as text file in a determined form. Text is stored in a ***text*** variable in ***tpc*** structure in a requested form. We checked the path for all possible directories by traversing on objects. When we found the path, related ***text*** variable of object is send to ***memcpy*** function as an parameter. Therefore, requested file is read. Read operation is similar to our previous project implementing a character devices. Our tpc\_read function and its detailed explanations with comment lines are given below:

static int **tpc\_read**(const char \*path, char \*buf, size\_t size, off\_t offset,

struct fuse\_file\_info \*fi) {

int i = 0;

char \*tempText;// is used when the path is found

tempText = (char \*) malloc(sizeof(char) \* 200);

memset(tempText, 0, sizeof(char) \* 200);

/\* strategy is similar to readdir function

\*/

while(i<linenumber){

/\*traverse on all object and create required varibles and

form all possible directories for each iteration \*/

char \*districtdir;

districtdir = (char \*) malloc(sizeof(char) \* 50);

memset(districtdir, 0, sizeof(char) \* 50);

char \*citydir;

citydir = (char \*) malloc(sizeof(char) \* 30);

memset(citydir, 0, sizeof(char) \* 30);

char \*neighborhooddir;

neighborhooddir = (char \*) malloc(sizeof(char) \* 60);

memset(neighborhooddir, 0, sizeof(char) \* 60);

strcat(citydir, folder\_path);

strcat(citydir, "/");

strcat(citydir, objs[i].city);

strcat(districtdir, citydir);

strcat(districtdir, "/");

strcat(districtdir, objs[i].district);

strcat(neighborhooddir, districtdir);

strcat(neighborhooddir, "/");

strcat(neighborhooddir, objs[i].neighborhood);

//EXAMPLE: "/NAMES/Istanbul/Sariyer/Maslak"

char \*cityCodedir;

cityCodedir = (char \*) malloc(sizeof(char) \* 10);

memset(cityCodedir, 0, sizeof(char) \* 10);

char \*codedir;

codedir = (char \*) malloc(sizeof(char) \* 18);

memset(codedir, 0, sizeof(char) \* 18);

strcat(cityCodedir, folder\_path2);

strcat(cityCodedir, "/");

strcat(cityCodedir, objs[i].cityCode);

strcat(codedir, cityCodedir);

strcat(codedir, "/");

strcat(codedir, objs[i].code);

//EXAMPLE: "/CODES/34/34398"

//There are only two cases for files

if (strcmp(path, neighborhooddir) == 0 ) {

strcat(tempText,objs[i].text);

break;

}

else if(strcmp(path, codedir) == 0)

{

strcat(tempText,objs[i].text);

break;

}

i++;

}

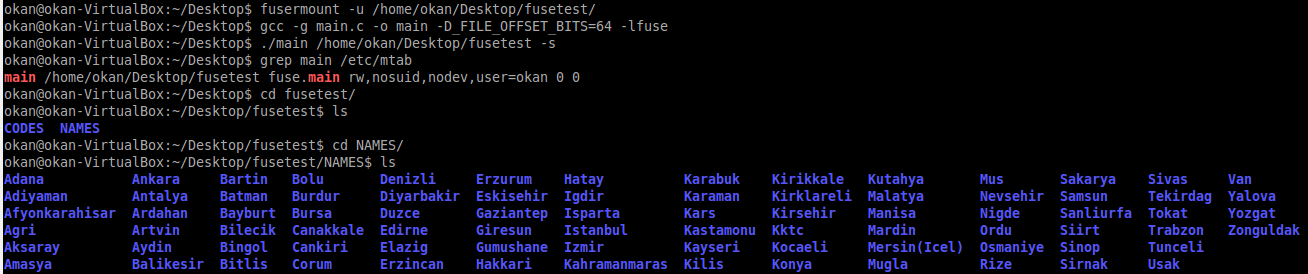
//for reading this text we are copying these content to buffer by using memcpy

//start from the offset and read up to size

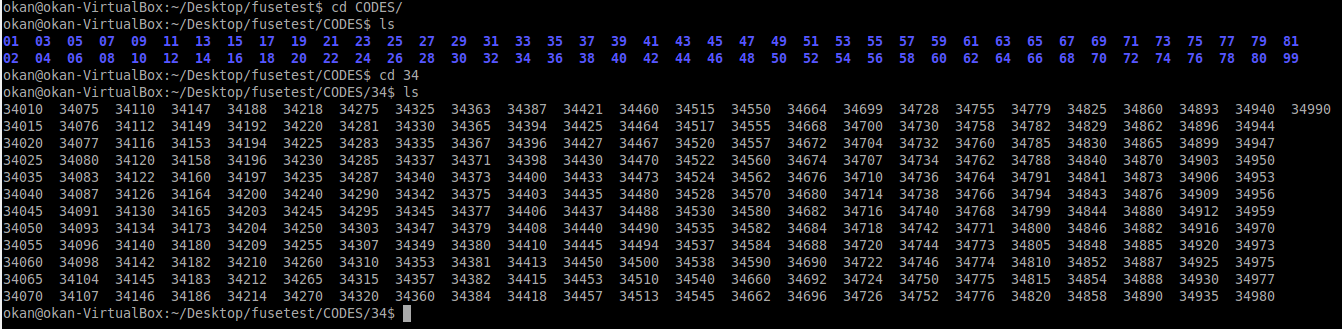
memcpy(buf, tempText + offset, size);

return strlen(tempText) - offset; // return number of bytes have been read

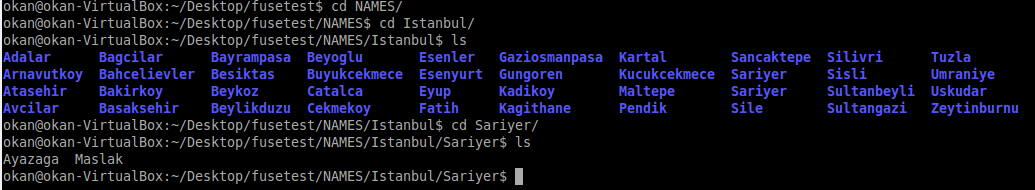
}

Consequently, we achieved the success to implement a FUSE file system which is constructed as an correct hierarchy and features according to project description. However, we could finish on time for doing delete and rename operations unfortunately is spite of the fact that we had started to implement project early. For us, it have been took so much time to wrote ***tpc\_readdir*** function. After submission we did some optimizations on project, yet we demonstrate original project at the demo session. For test part, some screenshots are given Figure 1- Figure 5 below:

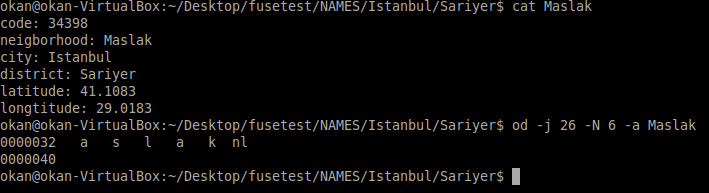
F*igure 1 Compiling, running and displaying directories in root and NAMES*



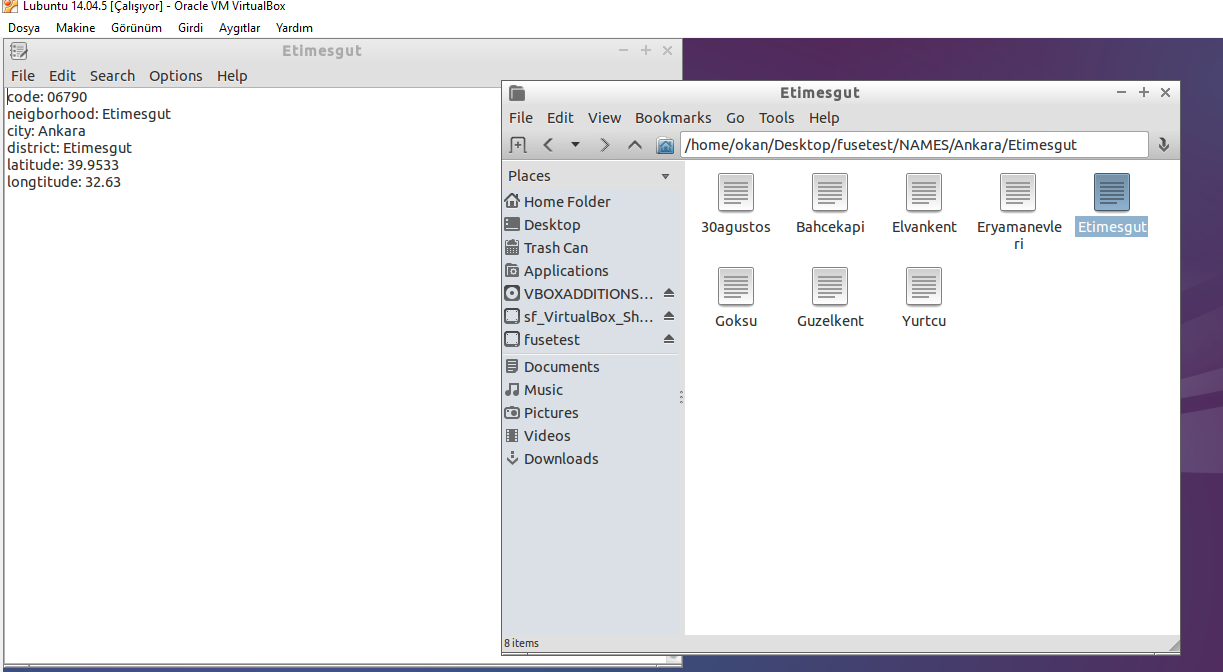
*Figure 2 Displaying directories in CODES*



*Figure 3 Displaying directories in Istanbul and files in Sariyer*



*Figure 4 Reading Maslak file and using some standart commands*



*Figure 5 Example on file manager*

1. **Conclusion**

It is inevitable fact that this project beneficial in terms of many aspect such as understanding the file system in OS , even implementing a file system by using FUSE and learning the distinction between working on user-space and kernel space. We faced many problems throughout the construction of the project, yet we overcome many of them and this way we developed our problem solving skills. I wish I was able to accomplish to complete the project with implementing rename and delete part.