DIRECTORY HISTORY MANAGER

Done By :

Tulika Somani – 190905482

Angad Sandhu – 190905494

Rasesh Rajpopat – 190905500

Mohammad Danish Eqbal - 190905513

DIRECTORY HISTORY MANAGER

A Mini-Project Report

Submitted by

Tulika Somani – 190905482

Angad Sandhu – 190905494

Rasesh Rajpopat – 190905500

Mohammad Danish Eqbal – 190905513

In partial fulfilment for the award of the degree of

Bachelor of Technology (B.Tech)

IN

Computer Science & Engineering

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Department of Computer Science & Engineering

December 2021

STUDENT DETAILS

1. Tulika Somani — Reg No: 190905482
   Roll No: 59 — Section: CSE – A

2. Angad Sandhu — Reg No: 190905494
   Roll No: 60 — Section: CSE – A

3. Rasesh Rajpopat — Reg No: 190905500
   Roll No: 61 — Section: CSE – A

4. Mohammad Danish Eqbal — Reg No: 190905482
   Roll No: 62 — Section: CSE – A

ABSTRACT

The file system and directory structure are a vital part of any Operating System. A file system contains thousands and millions of files, owned by several users. A directory is a container that is used to contain folders and files, and organize them in a hierarchical manner.

The idea of the project is to create text files pertaining to the directory structure at any point of time, and comparing them to denote the changes made in the directories and files.

Programming Language Used: C

CONTRIBUTION BY EACH STUDENT

Tulika Somani (59)

- Implemented the code which compares two file stamps to display any changes made in the directory structure. Any files added, removed or modified are reflected in the output.

Angad Sandhu (60)

- Implemented the code which creates file stamps of a particular directory. The file stamp displays the structure of the directory and any subdirectories present inside.

Rasesh Rajpopat (61)

- Implemented the code which compares two file stamps to display any changes made in the directory structure. Any files added, removed or modified are reflected in the output.

Mohammad Danish Eqbal (62)

- Implemented the code which creates file stamps of a particular directory. The file stamp displays the structure of the directory and any subdirectories present inside.

IMPLEMENTATION

CODE:

//creating file stamps

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

#include<time.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>

#include<sys/stat.h>
#include<sys/types.h>
#include <sys/time.h>

FILE *fptr;
char dateTodayFile[50];


/*
-t--12738348
-d--dir1
    -f--file1
    -d--dir2
        -f---
*/


// function to traverse the directory
void function(char* dir, int depth){

    // initialising our variables or handling files in LINUX
    DIR* d; struct
    dirent* item;
    struct stat mystat;
```

```
    // check to see if directory exists
    if((d=opendir(dir))==NULL){
        printf("Directory isn't opening: %s\n", dir);
        return;
    }

    // changing our cwd to the location mentioned in the function
    chdir(dir);

    // iterating through all items in our directory
    while((item = readdir(d))!=NULL){

        // getting a storing permission in mystat
        lstat(item->d_name,&mystat);

        // checking to see if we have a file or directory
        if(S_ISDIR(mystat.st_mode)){

            // for files
            if(strcmp(".",item->d_name)==0 || strcmp("..",item->d_name)==0 ||
strncmp(item->d_name, "filestamps", 3) == 0)
                continue;

            // for dirs by recursive call
            fprintf(fptr, "-d--%s,%d\n", item->d_name, depth);


            function(item->d_name,depth+4);

        } else {
            fprintf(fptr, "-f--%s,%d\n", item->d_name, depth);
        }
    }

    // going back to upper dir
    chdir("..");
    closedir(d);
}

void makedir(char dir[]){
    struct stat st = {0};

    strcat(dir, "/filestamps/");

    if (stat(dir, &st) == -1) {
        mkdir(dir, 0700);
    }
}
```

```c
void fileDateToday(){
    // get date
    time_t time_raw_format;
    struct tm * ptr_time;
    char buffer[100];

    time ( &time_raw_format );
    ptr_time = localtime ( &time_raw_format );

    if(strftime(buffer,100,"%Y_%m_%d__",ptr_time) == 0){
        perror("Couldn't prepare formatted string");
    }

    // get time
    time_t seconds;
    struct tm *timeStruct;
    seconds = time(NULL);
    timeStruct = localtime(&seconds);

    char tempHr[10], tempMin[10], tempSec[10];
    char timestr[50];
    bzero(timestr, sizeof(timestr));

    sprintf(tempHr, "%d_", timeStruct->tm_hour);
    strcat(timestr, tempHr);

    sprintf(tempMin, "%d_", timeStruct->tm_min);
    strcat(timestr, tempMin);
    sprintf(tempSec, "%d.", timeStruct->tm_sec);
    strcat(timestr, tempSec);
    strcat(timestr, "dat");

    // concatenating timestr and buffer
    strcat(buffer, timestr);

    // copying buffer to pointer
    strcpy(dateTodayFile, buffer);
}

void makeFile(char dir[], char dateTodayFile[]){
    strcat(dir, dateTodayFile);

   // use appropriate location if you are using MacOS or Linux
   fptr = fopen(dir,"w+");

   if(fptr == NULL)
   {
      printf("Error!\n");
```

```
      exit(1);
   }
}

void timestamp(){
    time_t seconds;

    seconds = time(NULL);

    fprintf(fptr, "-t--%ld,0\n", seconds);
}

// DRIVER CODE
int main(int argc, char *argv[]){

    if(argc <= 1){
        printf("The corect way of use is [executable] [directory]");
        exit(0);
    }

    // getting root file directory name
    char dir[100], dirt[100];
    strcpy(dir, argv[1]);
    strcpy(dirt, argv[1]);

    // making 'file stamps' directory if doesn't exist
    makedir(dirt);

    // getting string of today's date
    fileDateToday();

    // making file in the '/file stamps' dir
    makeFile(dirt, dateTodayFile);

    // getting todays timestamp and adding to directory
    timestamp();

    // Listing our cwd and depth of traversal
    function(dir,0);

    printf("Scanning of Directory [%s] Completed!!\n\n", dir);

    // closing file descriptor and exiting
    fclose(fptr);
    exit(0);
}
```

Output:

```
-t--1638096665,0
-d--2,0
-f--q3.c,4
-f--q4.py,4
-f--q2.txt,4
-d--3,0
-f--hi.txt,4
-d--4,4
-f--iWish.c,8
-f--hello.c,8
-d--1,0
-f--q2.c,4
-f--q1.txt,4
```

```
-t--1638096702,0
-d--2,0
-f--q4.py,4
-f--q2.txt,4
-d--4,4
-f--iWish.c,8
-f--hello.c,8
-d--3,0
-f--hi.txt,4
-d--1,0
-f--q2.c,4
-f--q1.txt,4
```

Explanation:

The given output denotes the directory and file structure at a single point of time. The ‘-t--’ shows the timestamp, the ‘-d--' shows that it is a directory and the ‘-f--' shows that it is a file. The number following the file name denotes the depth, which is useful for signifying the hierarchical structure of the directories and files.

//comparing two file stamps

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<ctype.h>
#include<string.h>

int main(int argc, char const *argv[])
{
    if(argc != 3)
    {
        printf("Insufficient arguments\n");
        exit(0);
    }

    int f1,f2;
    f1 = open(argv[1],O_RDONLY);
    f2 = open(argv[2],O_RDONLY);

    char buf1[100],buf2[100];
    char a,b;
    int i = 0,j = 0;

    read(f1,&a,sizeof(a));
    read(f2,&b,sizeof(b));

//storing timestamp
    while (a != '\n' && b != '\n')
    {
        buf1[i] = a;
        buf2[i] = b;
        i++;
        read(f1,&a,sizeof(a));
        read(f2,&b,sizeof(b));
    }

    close(f1);
    close(f2);

    int old,new;
```

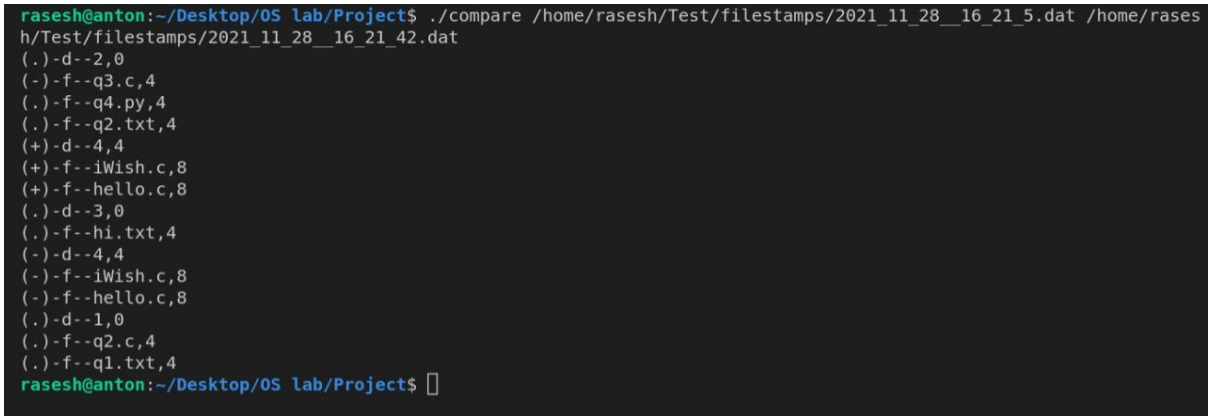
```
//comparing timestamp and assigning old, new accordingly
    int res = strcmp(buf1,buf2);
    if(res > 0)
    {
        old = open(argv[2],O_RDONLY);
        new = open(argv[1],O_RDONLY);
    }
    else
    {
        old = open(argv[1],O_RDONLY);
        new = open(argv[2],O_RDONLY);
    }

    int n = read(old,&a,sizeof(a));
    int m = read(new,&b,sizeof(b));

//ignore first line since it is timestamp
    while (a != '\n' && b != '\n')
    {
        n = read(old,&a,sizeof(a));
        m = read(new,&b,sizeof(b));
    }

    n = read(old,&a,sizeof(a));
    m = read(new,&b,sizeof(b));
    i = 0;
    j = 0;
    bzero(buf1,sizeof(buf1));
    bzero(buf2,sizeof(buf2));

//traversing file
    while(n != 0 && m != 0)
    {
//storing each line in buff
        while (a != '\n')
        {
            buf1[i++] = a;
            n = read(old,&a,sizeof(a));
        }
        buf1[i] = '\0';

        while(b != '\n')
        {
            buf2[j++] = b;
            m = read(new,&b,sizeof(b));
        }
        buf2[j] = '\0';
```

```
//if buff are equal, continue with next line in both files
        if(strcmp(buf1,buf2) == 0)
        {
            printf("(.)%s\n",buf1);
            bzero(buf1,sizeof(buf1));
            bzero(buf2,sizeof(buf2));
            n = read(old,&a,sizeof(a));
            m = read(new,&b,sizeof(b));
            i = 0;
            j = 0;
            continue;
        }
//buff are not equal
        else
        {
            char depth1 = buf1[strlen(buf1) - 1];
            char depth2 = buf2[strlen(buf2) - 1];
//depth is same, filename is not equal
            if (depth1 == depth2)
            {
                if(strcmp(buf1,buf2) > 0)
                {
                    printf("(+)%s\n",buf2);
                    int len = strlen(buf1);
                    lseek(old,-len-1,SEEK_CUR);

                    n = read(old,&a,sizeof(a));
                    m = read(new,&b,sizeof(b));
                    i = 0;
                    j = 0;
                    continue;
                }
                else if(strcmp(buf1,buf2) < 0)
                {
                    printf("(-)%s\n",buf1);
                    int len = strlen(buf2);
                    lseek(new,-len-1,SEEK_CUR);

                    n = read(old,&a,sizeof(a));
                    m = read(new,&b,sizeof(b));
                    i = 0;
                    j = 0;
                    continue;
                }
            }
```

```
//depth of old is smaller than new, files have been added
            else if(depth1 < depth2)
            {
                printf("(+)%s\n",buf2);
                int len = strlen(buf1);
                lseek(old,-len-1,SEEK_CUR);

                n = read(old,&a,sizeof(a));
                m = read(new,&b,sizeof(b));
                i = 0;
                j = 0;
                continue;
            }
//depth of new is smaller than old, files have been deleted
            else
            {
                printf("(-)%s\n",buf1);
                int len = strlen(buf2);
                lseek(new,-len-1,SEEK_CUR);

                n = read(old,&a,sizeof(a));
                m = read(new,&b,sizeof(b));
                i = 0;
                j = 0;
                continue;
            }
        }
    }

//if old file reaches EOF first, then remaining in new file have been added
    if(n == 0)
    {
        lseek(new,-1,SEEK_CUR);
        while(m != 0)
        {
            m = read(new,&b,sizeof(b));
            while(b != '\n')
            {
                buf2[j++] = b;
                m = read(new,&b,sizeof(b));
            }
            buf2[j] = '\0';
            if(m == 0)
                break;
            printf("(+)%s\n",buf2);
            bzero(buf2,sizeof(buf2));
            j = 0;
```

```
        }
    }

//if new file reaches EOF first, then remaining in old file have been removed
    if(m == 0)
    {
        lseek(old,-1,SEEK_CUR);
        while(n != 0)
        {
            n = read(old,&a,sizeof(a));
            while(a != '\n')
            {
                buf1[i++] = a;
                n = read(old,&a,sizeof(a));
            }
            buf1[i] = '\0';
            if(n == 0)
                break;
            printf("(-)%s\n",buf1);
            bzero(buf1,sizeof(buf1));
            i = 0;
        }
    }
}
```

Output:



Explanation:

Then (.) denotes that the directory/file has remained the same, the (-) denotes that the directory/file has been deleted and the (+) denotes that the directory/file has been added to the given directory.

LEARNING OUTCOME

We understood the intimate details related to the working of a file management system in LINUX. We got insights on how the directories and files are stored and managed by the Operating System. The various parsing techniques which are used by the compiler have been analysed while working on this project. Interaction of the Operating System with the hardware, user processes and file systems were also closely observed.

FUTURE SCOPE

The above basic implementation of Directory History Manager can be further more modified to display a detailed output with more statistical data. Number of lines in each file can be shown, if there were any modifications made to the file besides adding or removing them from the directory, then the line at which the modification took place inside a particular file can also be reflected in the output.

REFERENCES

[Gagne, Silberschatz and Galvin-9th Edition] Operating System Concepts

[Neil Matthew, Richard Stones – 4th Edition] Beginning Linux Programming

LINKS

1. Operating System | Structures of Directory - Tutorialspoint.dev
2. Linux Directory Structure - GeeksforGeeks
3. How to Find Difference Between Two Directories Using Diff and Meld Tools (tecmint.com)