Customised Virtual File System

-This project is used to emulate all functionalities provided by file systems.

Technology used

System Programming using C

User Interface

Command User Interface

Platform Required

Windows NT Platform OR Linux Distributions

Hardware Requirements

4CORE CPU, RAM,150GB

Description of The Project

1) In this project we emulate all data structure which are used by operating system to manage file system-oriented task.

2) As the name suggest its virtual because we maintain all the record in primary storage.

3) In this project we create all data structure which require for File subsystems as Inode,Inode Table,File Table,UAREA,User File Descriptor Table, Super Block, Disk Inode List Block, Data Block, Boot Block etc.

4) We provide all implementation of necessary system call and command of file subsystem as Open,Close,Read,Write,Lseek,Create,RM,LS,Stat,Fstat etc.

5) While providing the implementations of all above functionality we use our own data structure by referring Algorithms of UNNIX operating systems.

6) By using this project, we can get overview of UFS (Unix File System) on any platform.

Datastructure used in the project

Singly Linear

Singly Circular

Doubly Linear

Doubly Circular

Digram of Datasrtucture used in the project

Actual Code of The Project

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<iostream>

//#include<io.h>

#define MAXINODE 50

#define READ 1

#define WRITE 2

#define MAXIFILESIZE 2048

#define REGULAR 1

#define SPECIAL 2

#define START 0

#define CURRENT 1

#define END 2

typedef struct SuperBlock

{

    int TotalInodes;

    int FreeInode;

}SUPERBLOCK, \*PSUPERBLOCK;

typedef struct inode

{

    char FileName[50];

    int InodeNumber;

    int FileSize;

    int FileActualSize;

    int FileType;

    char \*Buffer;

    int LinkCount;

    int ReferenceCount;

    int permission;//1  23

    struct inode \* next;

}INODE,\*PINODE,\*\*PPINODE;

typedef struct filetable

{

    int readoffset;

    int writeoffset;

    int count;

    int mode;

    PINODE ptrinode;

}FILETABLE, \*PFILETABLE;

typedef struct ufdt

{

    PFILETABLE ptrfiletable;

}UFDT;

UFDT UFDTArr[50];

SUPERBLOCK SUPERBLOCKobj;

PINODE head = NULL;

void man(char \* name)

{

    if(name == NULL) return;

    if(strcmp(name,"create") == 0)

    {

        printf("Description : Used to create new regular file\n");

        printf("Usage : create File\_name permission\n");

    }

    else if(strcmp(name,"read") == 0)

    {

        printf("Description : is used to data from regular file\n");

        printf("Usage : read File\_name No\_Of\_Bytes\_To\_Read\n");

    }

    else if(strcmp(name,"write") == 0)

    {

        printf("Description : is used to write into regular file\n");

        printf("Usage : write File\_name\n After this enter the data that we want to write\n");

    }

    else if(strcmp(name,"ls") == 0)

    {

        printf("Description : is used to list all information into file\n");

        printf("Usage : ls\n");

    }

    else if(strcmp(name,"stat") == 0)

    {

        printf("Description : is used to display information of file\n");

        printf("Usage : stat File\_name\n");

    }

    else if(strcmp(name,"fstat") == 0)

    {

        printf("Description : is used to display information of file\n");

        printf("Usage : stat File\_Descriptor\n");

    }

    else if(strcmp(name,"truncate") == 0)

    {

        printf("Description : is used to remove data from file\n");

        printf("Usage : truncate File\_name\n");

    }

    else if(strcmp(name,"open") == 0)

    {

        printf("Descreption : is used to open existing file\n");

        printf("Usage : open File\_name mode\n");

    }

    else if(strcmp(name,"close") == 0)

    {

        printf("Descreption : is used to close opened file\n");

        printf("Usage : close File\_name\n");

    }

    else if(strcmp(name,"closeall") == 0)

    {

        printf("Descreption : is used to close all opened file\n");

        printf("Usage : closeall\n");

    }

    else if(strcmp(name,"lseek") == 0)

    {

        printf("Descreption : is used to change file offset\n");

        printf("Usage : lseek File\_name changeInOffset StartPoint\n");

    }

    else if(strcmp(name,"rm") == 0)

    {

        printf("Descreption : is used to delete the file\n");

        printf("Usage : rm File\_name\n");

    }

    else

    {

        printf("ERROR : No manual entry available.\n");

    }

}

void DisplayHelp()

{

    printf("ls : To List out all files\n");

    printf("clear :To clare console\n");

    printf("open : To open the file\n");

    printf("close : To close the file\n");

    printf("closeall : To close all opened files\n");

    printf("read : To read the content from file\n");

    printf("write : To write content into file\n");

    printf("exit : To Terminate file system\n");

    printf("stat : To Display information of file using name\n");

    printf("fstat : To Disply information of the file using file descriptor\n");

    printf("truncate : To remove all data from files\n");

    printf("rm : To Detect the file\n");

}

int GetFDFromName(char \* name)

{

    int i = 0;

    while(i < 50)

    {

        if(UFDTArr[i].ptrfiletable != NULL)

        {

            if(strcmp((UFDTArr[i].ptrfiletable->ptrinode->FileName),name) == 0)

            break;

        }

        i++;

    }

    if(i == 50)

    {

        return -1;

    }

    else

    {

        return i;

    }

}

PINODE Get\_Inode(char \* name)

{

    PINODE temp = head;

    int i = 0;

    if(name == NULL)

    {

        return NULL;

    }

    while(temp != NULL)

    {

        if(strcmp(name,temp->FileName) == 0)

        {

            break;

        }

        temp = temp -> next;

    }

    return temp;

}

void CreateDILB()

{

    int i = 1;

    PINODE newn = NULL;

    PINODE temp = head;

    while(i<= MAXINODE)

    {

            newn = (PINODE)malloc(sizeof(INODE));

        newn->LinkCount = 0;

        newn->ReferenceCount = 0;

        newn->FileType = 0;

        newn->FileSize = 0;

            newn->Buffer = NULL;

            newn->next = NULL;

        newn->InodeNumber = i;

            if(temp == NULL)

            {

                head = newn;

                temp = head;

            }

            else{

                temp->next = newn;

                temp=temp->next;

            }

            i++;

    }

    printf("DILB created successfully\n");

}

void InitialiseSuperBlock()

{

    int i = 0;

    while(i< MAXINODE)

    {

        UFDTArr[i].ptrfiletable = NULL;

        i++;

    }

    SUPERBLOCKobj.TotalInodes = MAXINODE;

    SUPERBLOCKobj.FreeInode = MAXINODE;

}

int CreateFile(char \*name,int permission)

{

    int i = 0;

    PINODE temp = head;

    if((name == NULL) || (permission == 0) || (permission > 3))

    {

        return -1;

    }

    if(SUPERBLOCKobj.FreeInode == 0)

    {

        return -2;

    }

    (SUPERBLOCKobj.FreeInode)--;

    if(Get\_Inode(name) != NULL)

    {

        return -3;

    }

    while(temp!=NULL)

    {

        if(temp->FileType == 0)

        {

            break;

        }

        temp = temp->next;

    }

    while(i<50)

    {

        if(UFDTArr[1].ptrfiletable == NULL)

        {

            break;

        }

        i++;

    }

    UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));

    UFDTArr[i].ptrfiletable->count = 1;

    UFDTArr[i].ptrfiletable->mode = permission;

    UFDTArr[i].ptrfiletable->readoffset = 0;

    UFDTArr[i].ptrfiletable->ptrinode = temp;

    strcpy(UFDTArr[i].ptrfiletable->ptrinode->FileName,name);

    UFDTArr[i].ptrfiletable->ptrinode->FileType = REGULAR;

    UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount = 1;

    UFDTArr[i].ptrfiletable->ptrinode->LinkCount = 1;

    UFDTArr[i].ptrfiletable->ptrinode->FileSize = MAXIFILESIZE;

    UFDTArr[i].ptrfiletable->ptrinode->FileActualSize = 0;

    UFDTArr[i].ptrfiletable->ptrinode->permission = permission;

    UFDTArr[i].ptrfiletable->ptrinode->Buffer = (char\*)malloc(MAXIFILESIZE);

    return i;

}

//rm\_File("Demo.txt")

int rm\_File(char \* name)

{

    int fd = 0;

    fd = GetFDFromName(name);

    if(fd == -1)

    {

        return -1;

    }

    (UFDTArr[fd].ptrfiletable->ptrinode->LinkCount)--;

    if(UFDTArr[fd].ptrfiletable->ptrinode->LinkCount == 0)

    {

        UFDTArr[fd].ptrfiletable->ptrinode->FileType = 0;

            //free(UFDTArr[fd].ptrfiletable->ptrinode->Buffer);

            free(UFDTArr[fd].ptrfiletable);

    }

        UFDTArr[fd].ptrfiletable = NULL;

        (SUPERBLOCKobj.FreeInode)++;

}

int ReadFile(int fd,char \* arr,int isize)

{

    int read\_size = 0;

    if(UFDTArr[fd].ptrfiletable == NULL)

    {

        return -1;

    }

    if(UFDTArr[fd].ptrfiletable->mode != READ && UFDTArr[fd].ptrfiletable->mode != READ + WRITE)

    {

        return -2;

    }

    if(UFDTArr[fd].ptrfiletable->ptrinode->permission != READ && UFDTArr[fd].ptrfiletable->ptrinode->permission != READ+WRITE)

    {

        return -2;

    }

    if(UFDTArr[fd].ptrfiletable->readoffset == UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)

    {

        return -3;

    }

    if(UFDTArr[fd].ptrfiletable->ptrinode->FileType != REGULAR)

    {

        return -4;

    }

    read\_size = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) - (UFDTArr[fd].ptrfiletable->readoffset);

    if(read\_size < isize)

    {

        strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->readoffset),read\_size);

        UFDTArr[fd].ptrfiletable->readoffset = UFDTArr[fd].ptrfiletable->readoffset + read\_size;

    }

    else

    {

        strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->readoffset),isize);

        UFDTArr[fd].ptrfiletable->readoffset = UFDTArr[fd].ptrfiletable->readoffset + isize;

    }

    return isize;

}

int WriteFile(int fd,char \* arr,int isize)

{

    if(((UFDTArr[fd].ptrfiletable->mode) != WRITE) && ((UFDTArr[fd].ptrfiletable->mode) != READ + WRITE))

    {

        return -1;

    }

    if(((UFDTArr[fd].ptrfiletable->ptrinode->permission) != WRITE) && ((UFDTArr[fd].ptrfiletable->ptrinode->permission) != READ + WRITE))

    {

        return -1;

    }

    if(UFDTArr[fd].ptrfiletable->writeoffset == MAXIFILESIZE)

    {

        return -2;

    }

    if(UFDTArr[fd].ptrfiletable->ptrinode->FileType != REGULAR)

    {

        return -3;

    }

    strncpy((UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable->writeoffset),arr,isize);

    (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) + isize;

    (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)+isize;

    return isize;

}

int OpenFile(char \*name,int mode)

{

    int i = 0;

    PINODE temp = NULL;

    if(name == NULL || mode <= 0)

    {

        return -1;

    }

    temp = Get\_Inode(name);

    if(temp == NULL)

    {

        return -2;

    }

    if(temp->permission < mode)

    {

        return -3;

    }

    while(i < 50)

    {

        if(UFDTArr[i].ptrfiletable == NULL)

        {

            break;

        }

        i++;

    }

    UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));

    if(UFDTArr[i].ptrfiletable == NULL)

    {

        return -1;

    }

    UFDTArr[i].ptrfiletable->count = 1;

    UFDTArr[i].ptrfiletable->mode = mode;

    if(mode == READ + WRITE)

    {

        UFDTArr[i].ptrfiletable->readoffset = 0;

        UFDTArr[i].ptrfiletable->writeoffset = 0;

    }

    else if(mode == READ)

    {

        UFDTArr[i].ptrfiletable->readoffset = 0;

    }

    else if(mode == WRITE)

    {

        UFDTArr[i].ptrfiletable->writeoffset = 0;

    }

    UFDTArr[i].ptrfiletable->ptrinode = temp;

    (UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)++;

    return i;

}

void CloseFileByName(int fd)

{

    UFDTArr[fd].ptrfiletable->readoffset = 0;

    UFDTArr[fd].ptrfiletable->writeoffset = 0;

    (UFDTArr[fd].ptrfiletable->ptrinode->ReferenceCount)--;

}

int CloseFileByName(char \*name)

{

    int i = 0;

    i = GetFDFromName(name);

    if(i == -1)

    {

        return -1;

    }

    UFDTArr[i].ptrfiletable->readoffset = 0;

    UFDTArr[i].ptrfiletable->writeoffset = 0;

    (UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)--;

    return 0;

}

void closeAllFile()

{

    int i = 0;

    while(i < 50)

    {

        if(UFDTArr[i].ptrfiletable != NULL)

        {

            UFDTArr[i].ptrfiletable->readoffset = 0;

            UFDTArr[i].ptrfiletable->writeoffset = 0;

            (UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)--;

        }

        i++;

    }

}

int LseekFile(int fd,int size,int from)

{

    if((fd < 0) || (from > 2))

    {

        return -1;

    }

    if(UFDTArr[fd].ptrfiletable == NULL)

    {

        return -1;

    }

    if((UFDTArr[fd].ptrfiletable->mode == READ) || (UFDTArr[fd].ptrfiletable->mode == READ + WRITE))

    {

        if(from == CURRENT)

        {

            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) > UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)

            {

                return -1;

            }

            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0)

            {

                return -1;

            }

            (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->readoffset) + size;

        }

        else if(from == START)

        {

            if(size > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))

            {

                return -1;

            }

            if(size < 0)

            {

                return -1;

            }

            (UFDTArr[fd].ptrfiletable->readoffset) = size;

        }

        else if(from == END)

        {

            if((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size > MAXIFILESIZE)

            {

                return-1;

            }

            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0)

            {

                return -1;

            }

            (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size;

        }

    }

    else if(UFDTArr[fd].ptrfiletable->mode == WRITE)

    {

        if(from == CURRENT)

        {

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > MAXIFILESIZE)

            {

                return -1;

            }

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0)

            {

                return -1;

            }

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))

            {

                (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->writeoffset) + size;

            }

            (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) + size;

        }

        else if(from == START)

        {

            if(size > MAXIFILESIZE)

            {

                return -1;

            }

            if(size < 0)

            {

                return -1;

            }

            if(size > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))

            {

                (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = size;

            }

            (UFDTArr[fd].ptrfiletable->writeoffset) = size;

        }

        else if(from == END)

        {

            if((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size > MAXIFILESIZE)

            {

                return-1;

            }

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0)

            {

                return -1;

            }

            (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size;

        }

    }

}

void ls\_file()

{

    int i = 0;

    PINODE temp = head;

    if(SUPERBLOCKobj.FreeInode == MAXINODE)

    {

        printf("Error : There are no files\n");

        return;

    }

    printf("\nFile Name\tInode number\tFile size\tLink count\n");

    printf("----------------------------------------------------------------\n");

    while(temp != NULL)

    {

        if(temp->FileType != 0)

        {

            printf("%s\t\t%d\t\t%d\t\t%d\n",temp->FileName,temp->InodeNumber,temp->FileActualSize,temp->LinkCount);

        }

        temp = temp->next;

    }

    printf("-----------------------------------------------------------------------------\n");

}

int fstat\_file(int fd)

{

    PINODE temp = head;

    int i = 0;

    if(fd < 0)

    {

        return -1;

    }

    if(UFDTArr[fd].ptrfiletable == NULL)

    {

        return -2;

    }

    temp = UFDTArr[fd].ptrfiletable->ptrinode;

    printf("\n----------Statical Information about file ----------\n");

    printf("FileName : %s\n",temp->FileName);

    printf("Inode Number %d\n",temp->InodeNumber);

    printf("FileSize : %d\n",temp->FileSize);

    printf("Actual File Size : %d\n,temp->FileActualSize");

    printf("Link count : %d\n,temp->LinkCount");

    printf("Referance count : %d\n",temp->ReferenceCount);

    if(temp->permission == 1)

    {

        printf("File permission : Read only\n");

    }

    else if(temp->permission == 2)

    {

        printf("File permission : write\n");

    }

    else if(temp->permission == 3)

    {

        printf("File permission : Read & Write\n");

    }

    printf("---------------------------------------------------------------\n\n");

    return 0;

}

int stat\_file(char\*name)

{

    PINODE temp = head;

    int i = 0;

    if(name == NULL)

    {

        return -1;

    }

    while(temp != NULL)

    {

        if(strcmp(name,temp->FileName) == 0)

        {

            break;

        }

        temp = temp->next;

    }

    if(temp == NULL)

    {

        return -2;

    }

    printf("\n----------Statical Information about file ----------\n");

    printf("FileName : %s\n",temp->FileName);

    printf("Inode Number %d\n",temp->InodeNumber);

    printf("FileSize : %d\n",temp->FileSize);

    printf("Actual File Size : %d\n,temp->FileActualSize");

    printf("Link count : %d\n,temp->LinkCount");

    printf("Referance count : %d\n",temp->ReferenceCount);

    if(temp->permission == 1)

    {

        printf("File permission : Read only\n");

    }

    else if(temp->permission == 2)

    {

        printf("File permission : write\n");

    }

    else if(temp->permission == 3)

    {

        printf("File permission : Read & Write\n");

    }

    printf("---------------------------------------------------------------------\n\n");

    return 0;

}

int truncate\_File(char \*name)

{

    int fd = GetFDFromName(name);

    if(fd == -1)

    {

        return -1;

    }

    memset(UFDTArr[fd].ptrfiletable->ptrinode->Buffer,0,1024);

    UFDTArr[fd].ptrfiletable->readoffset = 0;

    UFDTArr[fd].ptrfiletable->writeoffset = 0;

    UFDTArr[fd].ptrfiletable-> ptrinode->FileActualSize = 0;

}

int main()

{

    char \*ptr = NULL;

    int ret = 0,fd = 0,count = 0;

    char command[4][80],str[80],arr[1024];

    InitialiseSuperBlock();

    CreateDILB();

    while(1)

    {

        fflush(stdin);

        strcpy(str,"");

        printf("\nMarvellous VFS : >");

        fgets(str,80,stdin);

        count = sscanf(str,"%s%s%s%s",command[0],command[1],command[2],command[3]);

        if(count == 1)

        {

            if(strcmp(command[0],"ls") == 0)

            {

                ls\_file();

            }

            else if(strcmp(command[0],"closeall") == 0)

            {

                closeAllFile();

                printf("All files closed succesfully\n");

                continue;

            }

            else if(strcmp(command[0],"clear") == 0)

            {

                system("cls");

                continue;

            }

            else if(strcmp(command[0],"help") == 0)

            {

                DisplayHelp();

                continue;

            }

            else if(strcmp(command[0],"exit") == 0)

            {

                printf("Terminating the marvellous virtual file system\n");

                break;

            }

            else

            {

                printf("\nERROR : Command not found!!!\n");

                continue;

            }

        }

        else if(count == 2)

        {

            if(strcmp(command[0],"fstat") == 0)

            {

                ret = fstat\_file(atoi(command[1]));

                if(ret == -1)

                {

                    printf("ERROR : Incorrect parametrs\n");

                }

                if(ret == -2)

                {

                    printf("ERROR : There id no such files\n");

                }

                continue;

            }

            else if(strcmp(command[0],"fstat") == 0)

            {

                ret = fstat\_file(atoi(command[1]));

                if(ret == -1)

                {

                    printf("ERROR : Incorrect parametrs\n");

                }

                if(ret == -2)

                {

                    printf("ERROR : There is no such file\n");

                }

                continue;

            }

            else if(strcmp(command[0],"Close") == 0)

            {

                ret = CloseFileByName(command[1]);

                if(ret == -1)

                {

                    printf("ERROR : There is no such file\n");

                }

                continue;

            }

            else if(strcmp(command[0],"rm") == 0)

            {

                ret = rm\_File(command[1]);

                if(ret == -1)

                {

                    printf("ERROR : There is no such file\n");

                }

                continue;

            }

            else if(strcmp(command[0],"man") == 0)

            {

                man(command[1]);

            }

            else if(strcmp(command[0],"write") == 0)

            {

                fd = GetFDFromName(command[1]);

                if(fd == -1)

                {

                    printf("Error : Incorrect parametr\n");

                    continue;

                }

                printf("Enter the data : \n");

                scanf("%[^\n]",arr);

                ret = strlen(arr);

                if(ret == 0)

                {

                    printf("Error : Incorrect parametr\n");

                    continue;

                }

                ret = WriteFile(fd,arr,ret);

                if(ret == -1)

                {

                    printf("ERROR : Permission denied\n");

                }

                if(ret == -2)

                {

                    printf("ERROR : There is no suffiecnt memory to write \n");

                }

                if(ret == -3)

                {

                    printf("ERROR : It is not regular file\n");

                }

            }

            else if(strcmp(command[0],"truncate") == 0)

            {

                ret = truncate\_File(command[1]);

                if(ret == -1)

                {

                    printf("Error : Incorrect parameter\n");

                }

            }

            else

            {

                printf("\nERROR : Command not found !!!\n");

                continue;

            }

        }

        else if(count == 3)

        {

            if(strcmp(command[0],"create") == 0)

            {

                ret = CreateFile(command[1],atoi(command[2]));

                if(ret >= 0)

                {

                    printf("File is succesfully created with file descriptor : %d\n",ret);

                }

                if(ret == -1)

                {

                    printf("ERROR : Incorrect parametres\n");

                }

                if(ret == -2)

                {

                    printf("ERROR :File not present\n");

                }

                if(ret == -3)

                {

                    printf("ERROR : Permission denied\n");

                }

                continue;

            }

            else if(strcmp(command[0],"read") == 0)

            {

                ret = GetFDFromName(command[1]);

                if(fd == -1)

                {

                    printf("Error : Indirect parameter\n");

                    continue;

                }

                ptr = (char\*)malloc(sizeof(atoi(command[2]))+1);

                if(ptr == NULL)

                {

                    printf("Error : Memory allocation failuar\n");

                    continue;

                }

                ret = ReadFile(fd,ptr,atoi(command[2]));

                if(ret == -1)

                {

                    printf("ERROR : File not existing\n");

                }

                if(ret == -2)

                {

                    printf("ERROR : permission denied\n");

                }

                if(ret == -3)

                {

                    printf("ERROR : Reachrd at end of file\n");

                }

                if(ret == -4)

                {

                    printf("ERROR : It is not regular file \n");

                }

                if(ret == 0)

                {

                    printf("ERROR :File empty\n");

                }

                if(ret > 0)

                {

                    write(2,ptr,ret);

                }

                continue;

            }

            else

            {

                printf("\nERROR : Command not found !!!\n");

                continue;

            }

        }

        else if(count == 4)

        {

            if(strcmp(command[0],"|seek") == 0)

            {

                fd = GetFDFromName(command[1]);

                if(fd == -1)

                {

                    printf("ERROR : Incorrect parametrr\n");

                    continue;

                }

                ret = LseekFile(fd,atoi(command[2]),atoi(command[3]));

                if(ret == -1)

                {

                    printf("ERROR :Unable to perform lseek\n");

                }

            }

            else{

                printf("\nERROR : Command not found !!!\n");

                continue;

            }

        }

        else

        {

            printf("\nERROR : command not found!!!\n");

            continue;

        }

    }

    return 0;

}

Screenshot of output which demonstracte evry features