**Customised Vistual File System**

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

**Platform required**

Windows NT platform OR Linux Distributions

**Architectural requirement**

Intel 32 bit processor

**User Interface**

Command User Interface

**SDK used**

None

**Technology used**

System Programming using C

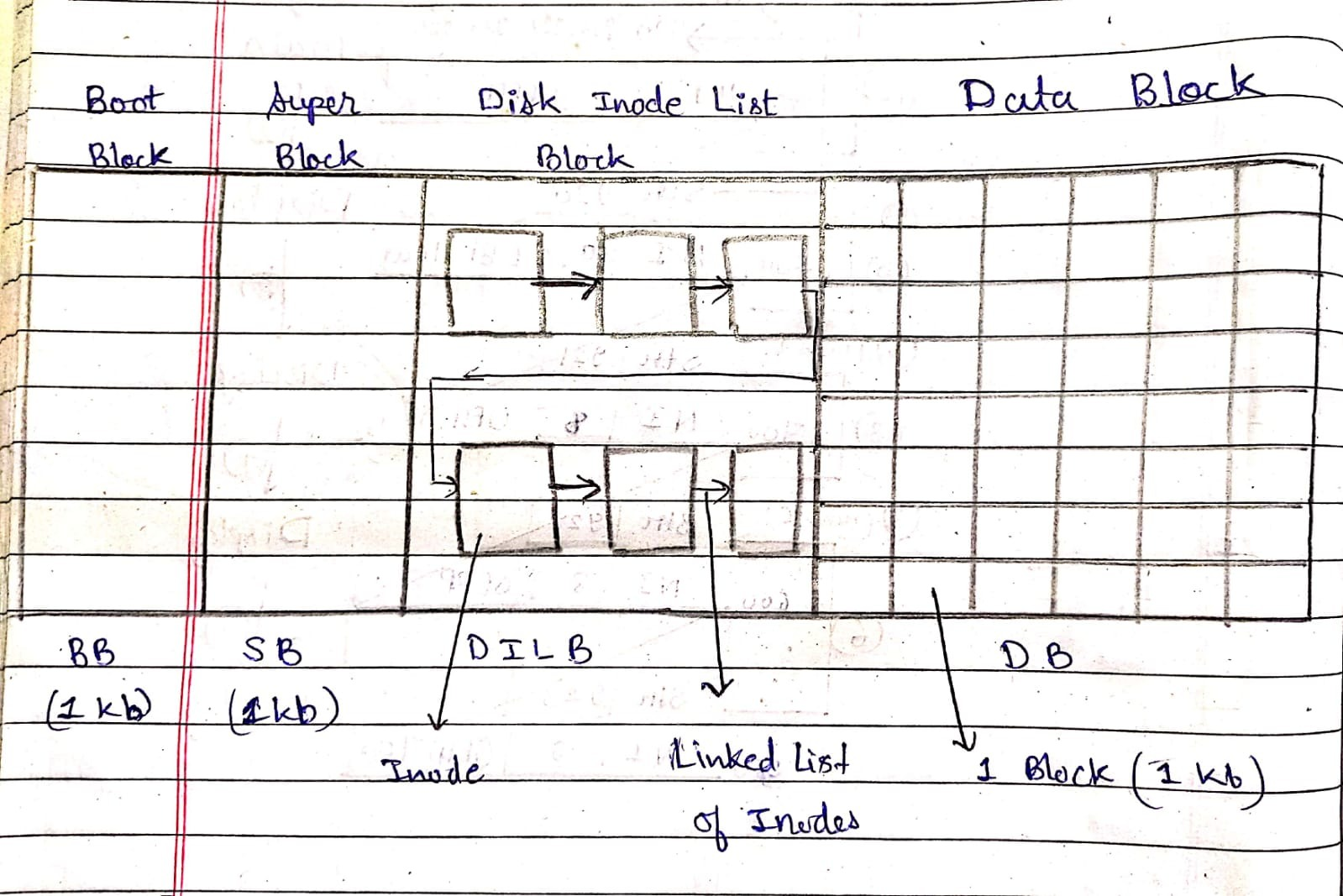
**About Virtual File System**

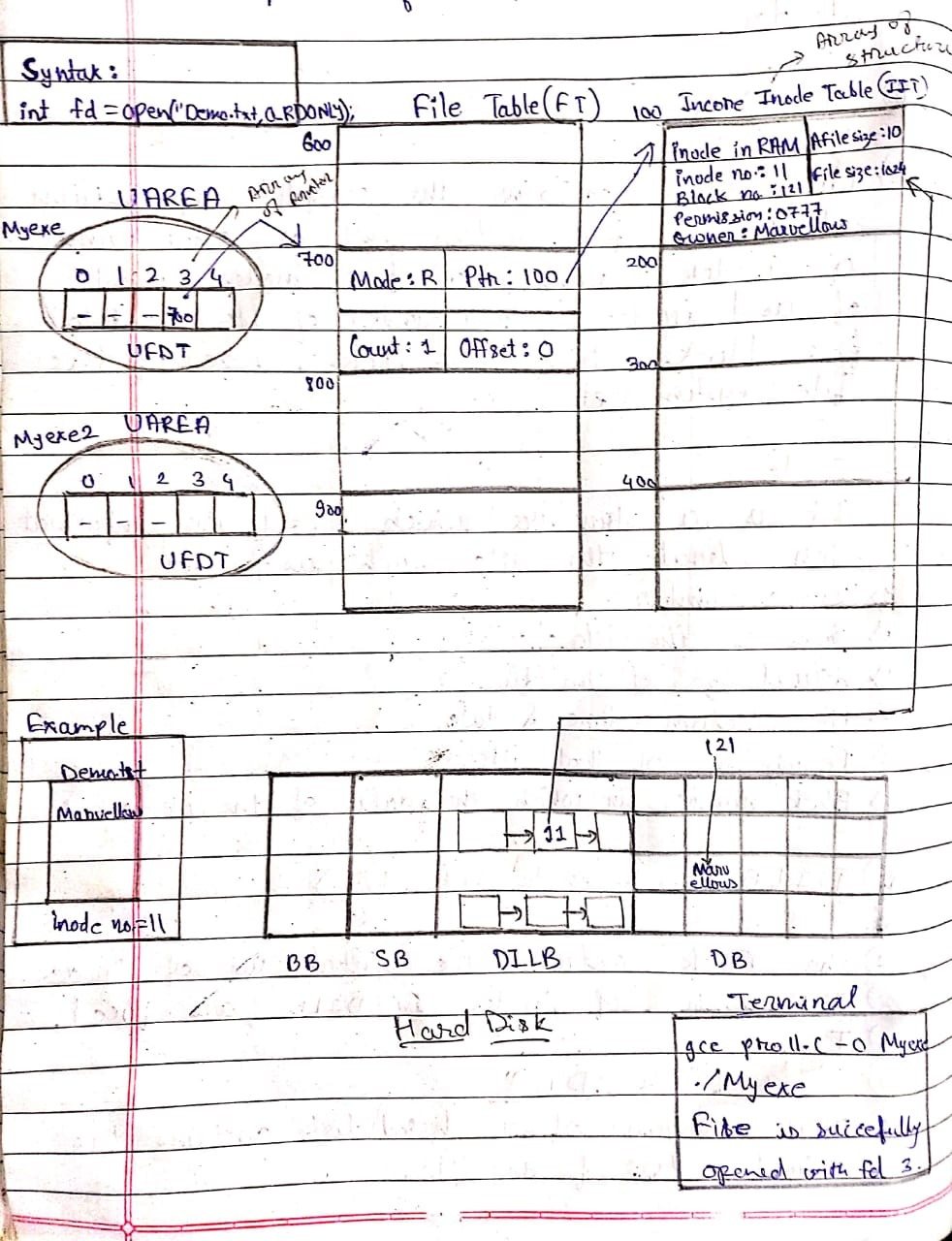
* In this project we emulate all data structures which are used by operating system to manage File system oriented tasks.
* As the name suggests its virtual because we maintain all records in primary storage.
* In this project we create all data structures which is required for File Subsystems as Inode, Inode Table, File Table, UAREA, User File Descriptor Table, Super block, Disk Inode List Block, Data Block, Boot Block etc.
* We provide all implementation of necessary system calls and commands of File subsystem as Open, Close, Read, Write, Lseek, Create, rm, ls, stat, fstat, etc.
* While providing the implementations of all above functionality we use our own data structures by referring Algorithms of UNNIX operating system.
* By using this project we can get overview of UFS (UNIX File System) on any platform.

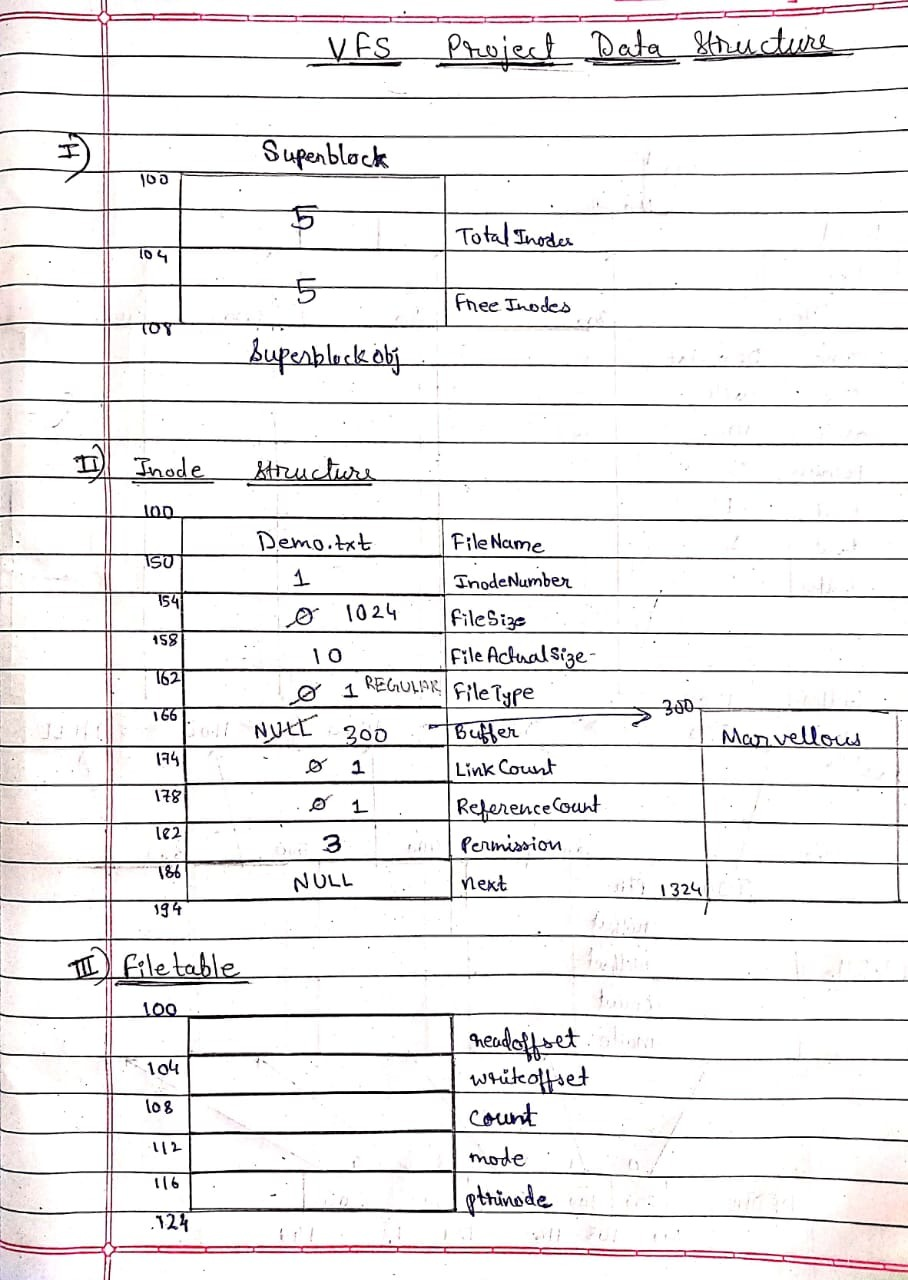
**Data Structure used in project**

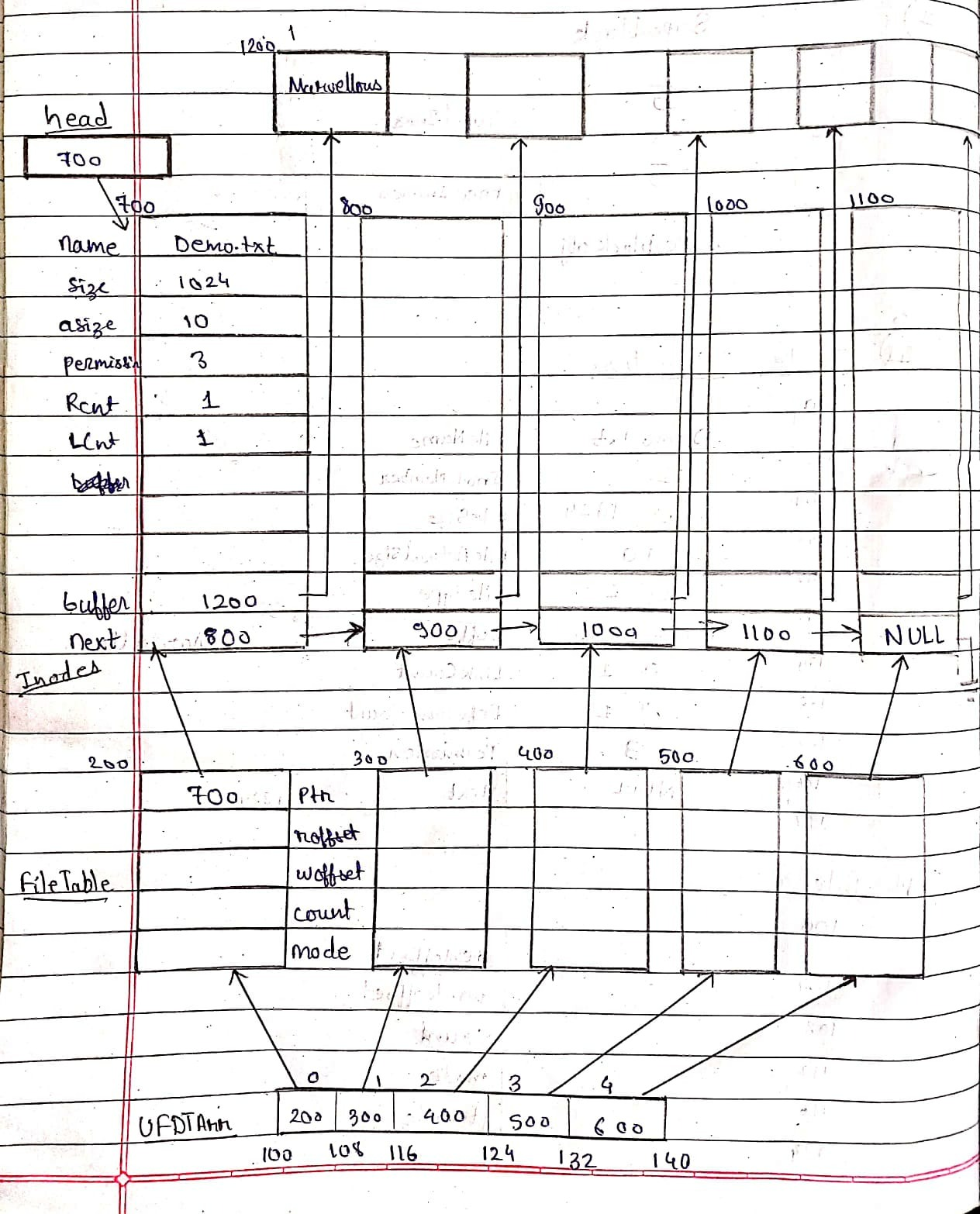
1. Structure.
2. Linked list.
3. Array.

**Diagrams -**









**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 MAXFILESIZE 2048

#define REGULAR 1

#define SPECIAL 2

#define START 0

#define CURRENT 1

#define END 2

typedef struct superblock

{

    int TotalInodes;

    int FreeInodes;

}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; // 1 2 3

    PINODE ptrinode;

}FILETABLE,\*PFILETABLE;

typedef struct ufdt

{

    PFILETABLE ptrfiletable;

}UFDT;

UFDT UFDTArr[MAXINODE];

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 : Used to read data from regular file\n");

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

    }

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

    {

        printf("Description : 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 : Used to list all information of file\n");

        printf("Usage : ls\n");

    }

    else if(strcmp(name,"stat") == 0)

    {

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

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

    }

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

    {

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

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

    }

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

    {

        printf("Description : Used to remove data from the file\n");

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

    }

    else if(strcmp(name,"open") == 0)

    {

        printf("Description : Used to open existing file\n");

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

    }

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

    {

        printf("Description : Used to close opened file\n");

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

    }

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

    {

        printf("Description : Used to close all opened files\n");

        printf("Usage : closeall\n");

    }

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

    {

        printf("Description : Used to change file offset\n");

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

    }

    else if(strcmp(name,"rm") == 0)

    {

        printf("Description : 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 clear 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 contents from file\n");

    printf("write : To Write contents into the file\n");

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

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

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

    printf("truncate : To Remove all data from file\n");

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

}

int GetFDFromName(char \*name)

{

    int i = 0;

    while(i<MAXINODE)

    {

        if((UFDTArr[i].ptrfiletable != NULL) && (UFDTArr[i].ptrfiletable->ptrinode->FileType != 0))

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

                break;

        i++;

    }

    if(i == MAXINODE)     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 succesfully\n");

}

void InitializeSuperBlock()

{

    int i = 0;

    while(i< MAXINODE)

    {

        UFDTArr[i].ptrfiletable = NULL;

        i++;

    }

    SUPERBLOCKobj.TotalInodes = MAXINODE;

    SUPERBLOCKobj.FreeInodes = MAXINODE;

}

int CreateFile(char \*name,int permission)

{

    int i = 0;

    PINODE temp = head;

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

        return -1;

    if(SUPERBLOCKobj.FreeInodes == 0)

        return -2;

    (SUPERBLOCKobj.FreeInodes)--;

    if(Get\_Inode(name) != NULL)

        return -3;

    while(temp != NULL)

    {

        if(temp->FileType == 0)

            break;

        temp = temp->next;

    }

    while(i<50)

    {

        if(UFDTArr[i].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->writeoffset = 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 = MAXFILESIZE;

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

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

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

    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.FreeInodes)++;

}

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 == MAXFILESIZE)      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)--;

            break;

        }

        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 > MAXFILESIZE)   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) >  MAXFILESIZE)   return -1;

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0) return -1;

            if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) >  UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)   return -1;

                (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 > MAXFILESIZE)    return -1;

            if(size < 0)    return -1;

            if(size > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))     return -1;

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

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

        }

        else if(from == END)

        {

            if((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size > MAXFILESIZE)   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.FreeInodes == MAXINODE)

    {

        printf("Error : There are no visible 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---------Statistical Information about the file----------\n");

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

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

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

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

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

    printf("Reference 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---------Statistical Information about the file----------\n");

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

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

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

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

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

    printf("Reference 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[MAXFILESIZE];

    InitializeSuperBlock();

    CreateDILB();

    while(1)

    {

        fflush(stdin);

        strcpy(str,"");

        printf("\nSP CVFS : > ");

        fgets(str,80,stdin); // scanf("%[^'\n']s",str);

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

        if(count == 1)

        {

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

            {

                ls\_file();

            }

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

            {

                CloseAllFile();

                printf("All files closed successfully\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 SP Virtual File System\n");

                break;

            }

            else

            {

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

                continue;

            }

        }

        else if(count == 2)

        {

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

            {

                ret = stat\_file(command[1]);

                if(ret == -1)

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

                if(ret == -2)

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

                continue;

            }

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

            {

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

                if(ret == -1)

                    printf("ERROR : Incorrect parameters\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 parameter\n");

                    continue;

                }

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

                scanf("%[^'\n']s",arr);

                ret = strlen(arr);

                if(ret == 0)

                {

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

                    continue;

                }

                ret = WriteFile(fd,arr,ret);

                if(ret == -1)

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

                if(ret == -2)

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

                if(ret == -3)

                    printf("ERROR : It is not a 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 successfully created with the file descriptor : %d\n",ret);

                if(ret == -1)

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

                if(ret == -2)

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

                if(ret == -3)

                    printf("ERROR : File already exists\n");

                if(ret == -4)

                    printf("ERROR : Memory allocation failure\n");

                continue;

            }

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

            {

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

                if(ret >= 0)

                    printf("File is successfully opened with the file descriptor : %d\n",ret);

                if(ret == -1)

                    printf("ERROR : Incorrect parameters\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)

            {

                fd = GetFDFromName(command[1]);

                if(fd == -1)

                {

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

                    continue;

                }

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

                if(ptr == NULL)

                {

                    printf("Error : Memory allocation failure\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 : Reached at the end of the file\n");

                if(ret == -4)

                    printf("ERROR : It is not a 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],"lseek") == 0)

            {

                fd = GetFDFromName(command[1]);

                if(fd == -1)

                {

                    printf("ERROR : Incorrect parameter\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;

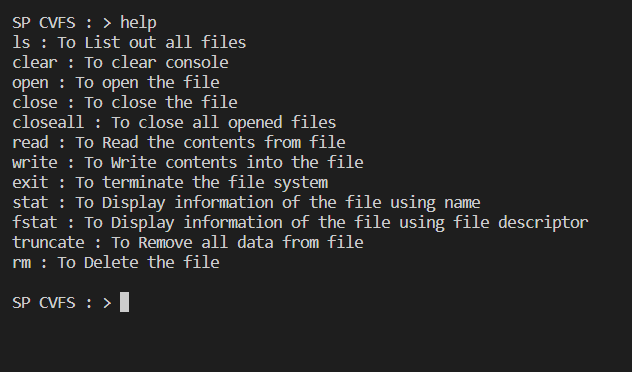
        }

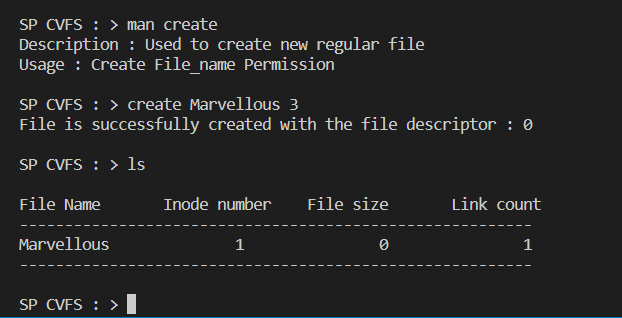
    }

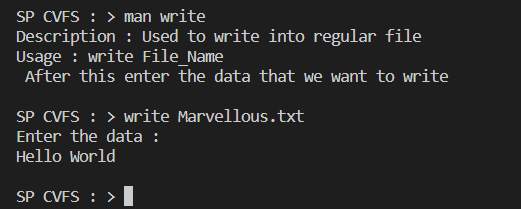
    return 0;

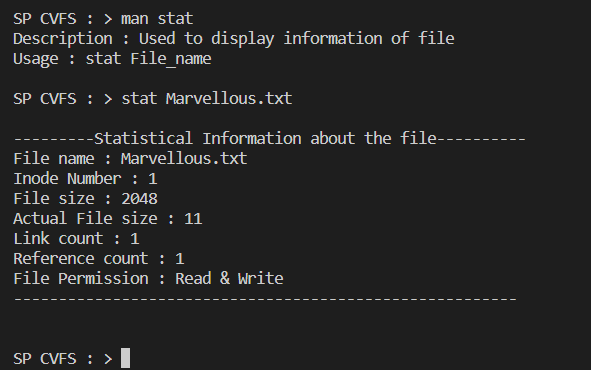
}

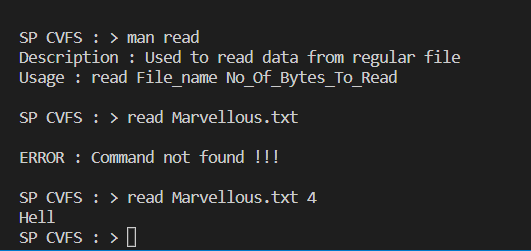
**Output**

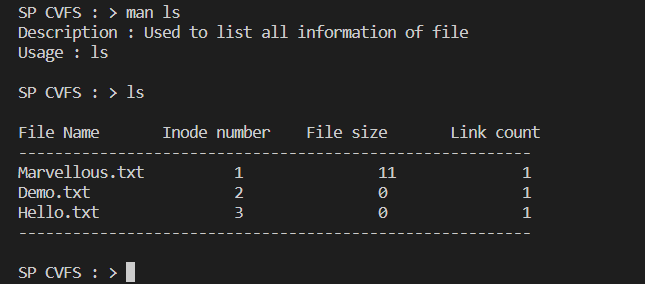
****

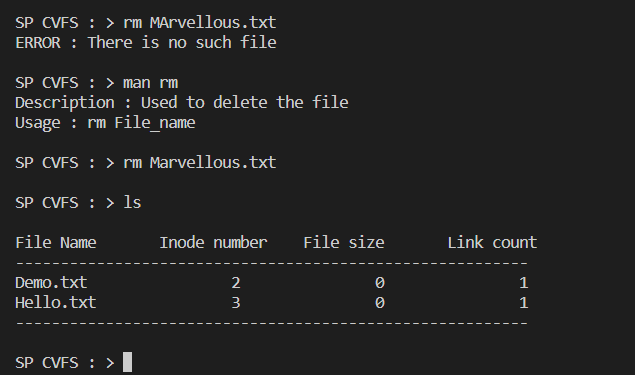
****

****

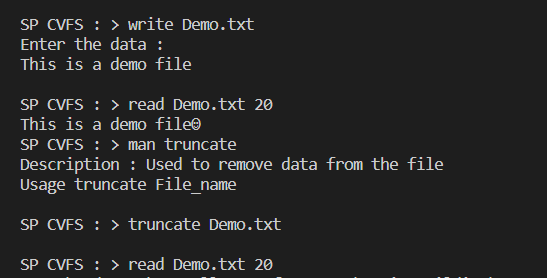
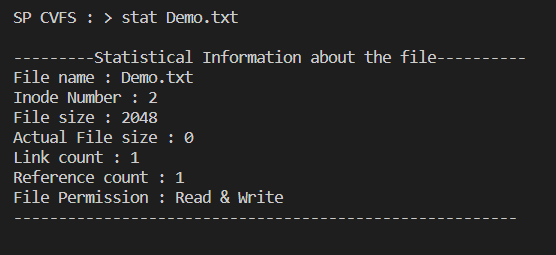
****

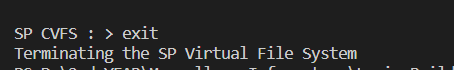
****

****

****

****

**** ****

****

**1 What is mean by file system?**

Ans: File system is a data structure used by operating system to control how data is stored and retirieved.

**2Which file systems are used by Linux and windows operating systems?**

Ans: Windows uses FAT(File Allocation Table) and NTFS(New Technology File System) as file systems, while Linux uses a variety of file systems such as UFS(UNIX File System), ext(Extended File Systems)

**3What are the parts of the file system?**

Ans :

1. **Explain UAREA and its contents.**

Ans : An operating system maintains a region called u-area i.e. user area which holds the specific information of process and stored in a stack segment.

The u-area of a process contains the following information

* A pointer in the process table identifies the entry that corresponds to the u-area.
* The real and effective user IDs determine various privileges allowed the process, such as file access rights.
* Timer field records the time the process spent executing in user mode and in kernel mode.
* An array indicates how the process wishes to react to signals.
* The control terminal field identifies the "login terminal" associated with the process if one exists.
* An error field records errors encountered during a system call.
* A return value field contains the result of system calls.
* I/O parameters describe the amount of data to transfer, the address of the source (or target) data array in user space, file offsets for I/O, and so on.
* The current directory and current root describe the file system environment of the process.
* The user file descriptor table records the files the process has open.
* Limit fields restrict the size of a process and the size of a file it can write.
* A permission modes field masks mode settings on files the process creates.

1. **Explain the use of the File Table and its contents.**

Ans :

1. **Explain the use of InCore inode Table and its use.**

Ans :

1. **What is mean by inode?**

Ans: It is a structure which holds the information about the file such as –

1. Inode number.
2. Size of the file.
3. Actual size of the file.
4. File creation time and date
5. Permission of that file.
6. Block number in which the data of the file gets stored.
7. **What are the contents of Superblock?**

Ans: Superblock contains the complete information about the file system such as-

1. Total number of inodes
2. Number of free inodes
3. Number of used inodes
4. Total number of blocks
5. Total free blocks
6. Total used blocks
7. Size of file system etc
8. **What are the types of files?**

Ans: There are two types of files-

1. Regular file.
2. Special file.
3. **What are the contents of the inode?**

Ans :

1. Inode number.
2. Size of the file.
3. Actual size of the file.
4. File creation time and date
5. Permission of that file.
6. Block number in which the data of the file gets stored.
7. **What is the use of a directory file?**

Ans : A directory file is used to store, organize, and separate files and directories on a computer.

1. **How the operating system maintains security for files?**

Ans : The most common techniques used to protect operating systems include the use of antivirus software and other endpoint protection measures, regular OS patch updates, a firewall for monitoring network traffic, and enforcement of secure access through least privileges and user controls.

1. **What happens when a user wants to open the file?**

Ans : when you open a file you are actually requesting the operating system to load the desired file ( copy the contents of file ) from the secondary storage to ram for processing

1. **What happens when a user calls lseek system call?**

Ans : **lseek()** system call repositions the read/write file offset i.e., it changes the positions of the read/write pointer within the file. In every file any read or write operations happen at the position pointed to by the pointer.

1. **What is the difference between library function and system call?**

Ans : A system call is a request made by the program to enter into kernel mode to access a process.

Whereas a library call is requwat made by the program to access a library function defined in a programming library.

1. **What is the use of this project?**

Ans :

1. **What are the difficulties that you faced in this project?**

Ans :

1. **Is there any improvement needed in this project?**

Ans :

**Explain the internal working of below system calls**

**(Write the solutions in the documentation)**

1. open
2. close
3. read
4. write
5. lseek
6. stat
7. chmod
8. unlink

Add a screenshot of each above command which demonstrates its use.

**Explain use of below commands**

**(Write the solutions in the documentation)**

1. ls

To list all the files of the current directory

1. ls – l

The -l option signifies the long list format.

1. ls – a

ls -a option flag lists all files including hidden files starting with '.'

1. rm

To delete a file

1. cat

cat command allows us to create single or multiple files, view content of a file, concatenate files and redirect output in terminal or files.

1. cd

cd command is used to change directory

1. chmod

 the chmod command is used to change the access mode of a file

1. cp

cp command copies files

1. df

The df command (short for disk free), is used to display information related to file systems about total space and available space.

1. find

used to find files and directories.

1. grep

used to search for a string of characters in a specified file.

1. ln

 ln command is used to create hard links and soft links for files in Linux.

1. mkdir

mkdir command in Linux allows the user to create directories

1. pwd

pwd stands for Print Working Directory. It prints the path of the working directory, starting from the root.

1. touch

The touch command creates files in Linux through the terminal.

1. uname

To display system information

1. stat

The stat is a command gives information about the file and filesystem.

1. man

The man command is a built-in manual for using Linux commands.

1. mkfs

The mkfs command makes file systems. On other operating systems, creating a file system is called formatting