Customized Virtual File System

Project Documentation

Name of the project - Customized Virtual File System

Technology used - System Programming using C

User Interface used - Command User Interface

Platform required - Windows NT platform / Linux Distributions

Architectural requirement - Intel 32 bit processor

Hardware requirements - None

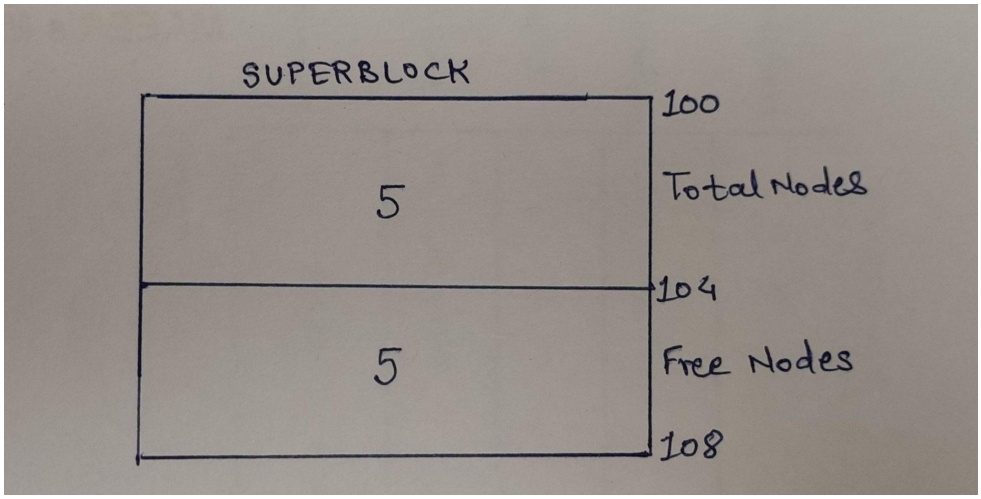
Data Structures used in the Project - LinkedList, Arrays

Description of the project -

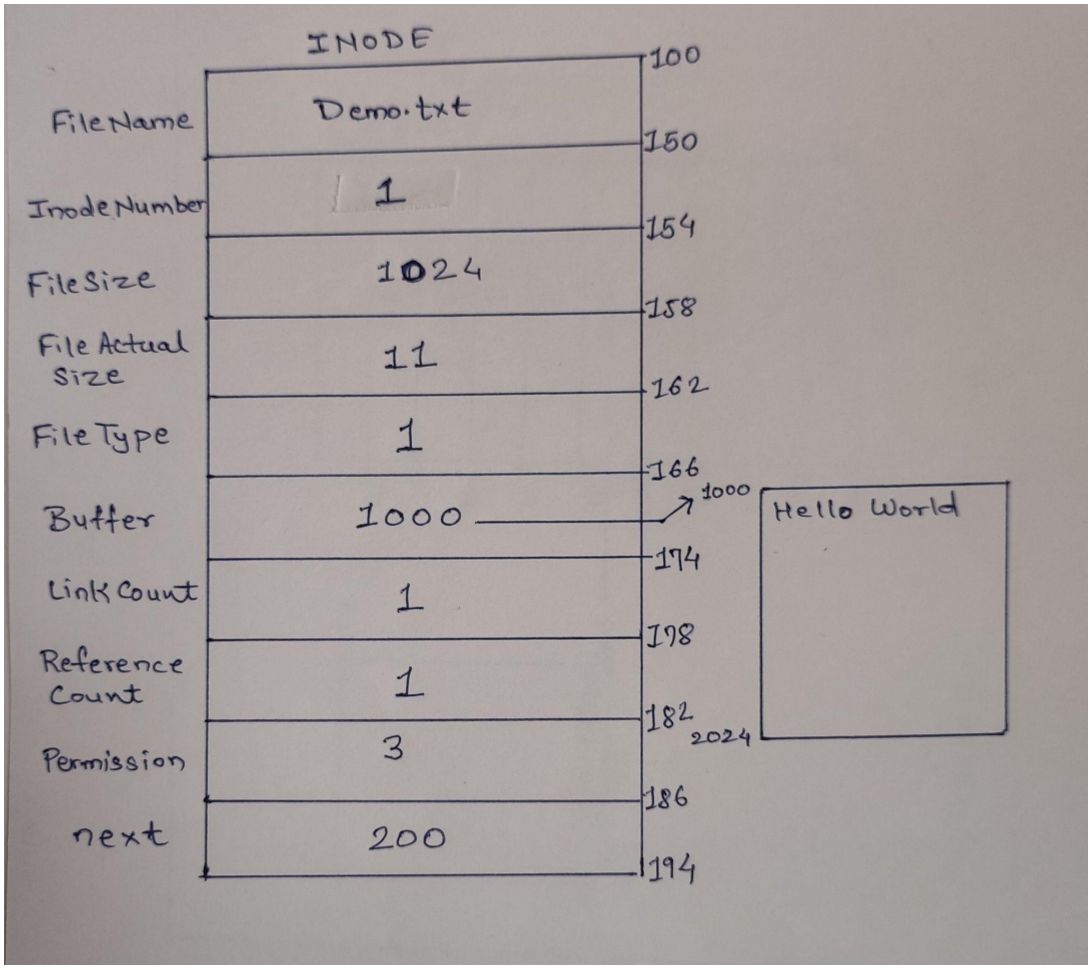
- This project is the virtual representation of a File system.
- File system is considered as the way of storing the information about the files and the data from the files in the secondary storage device.
- Every operating system has its own type of file system (NTFS / FAT- 32 / FAT - 16 / UFS).
- In this project we implement almost every system call used in final manipulation activity along with some important commands.
- This is a research based project and is developed to understand the internal concepts of an Operating System and to explore data structure implementation of C programming.

Diagrams of data structures used in the project

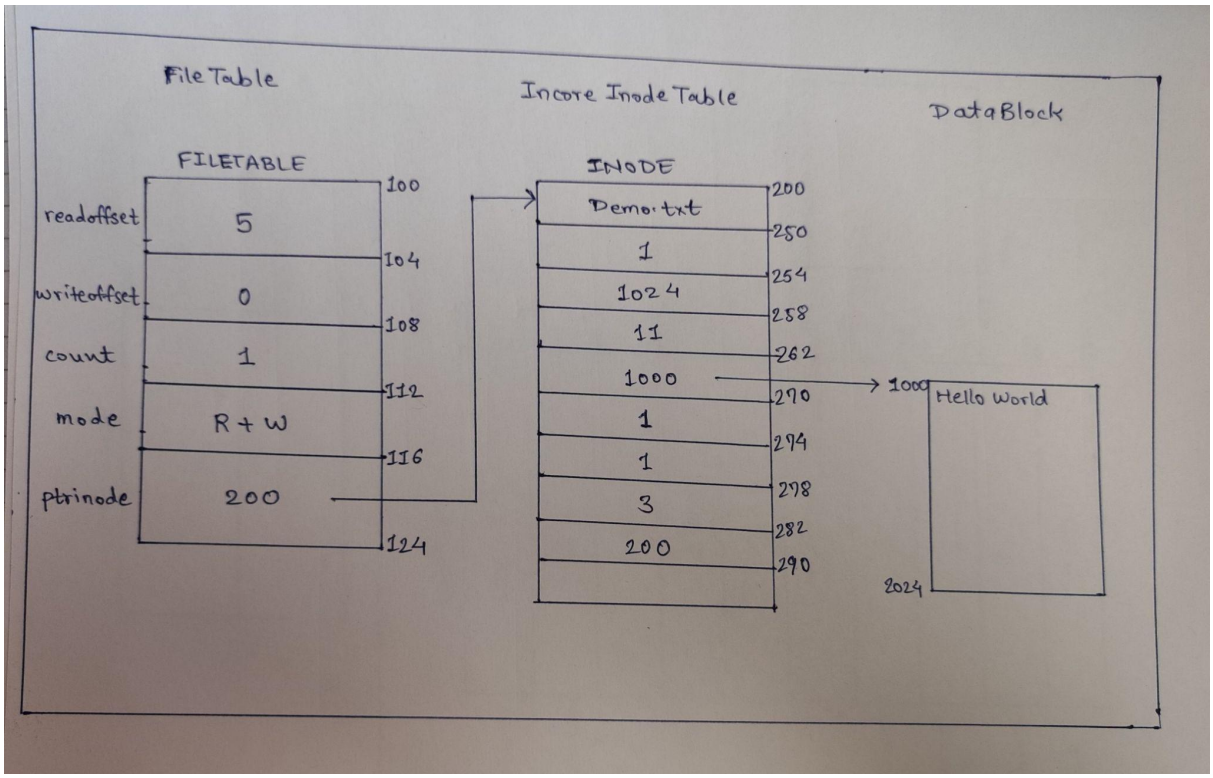
SUPERBLOCK -



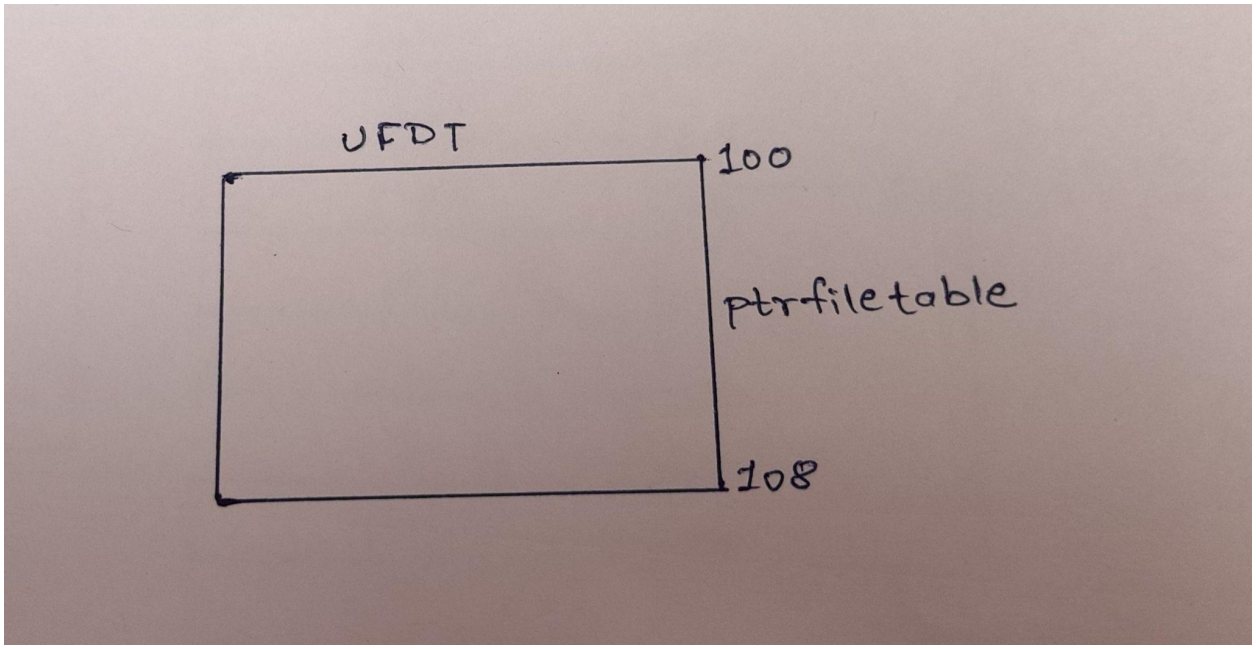
INODE -



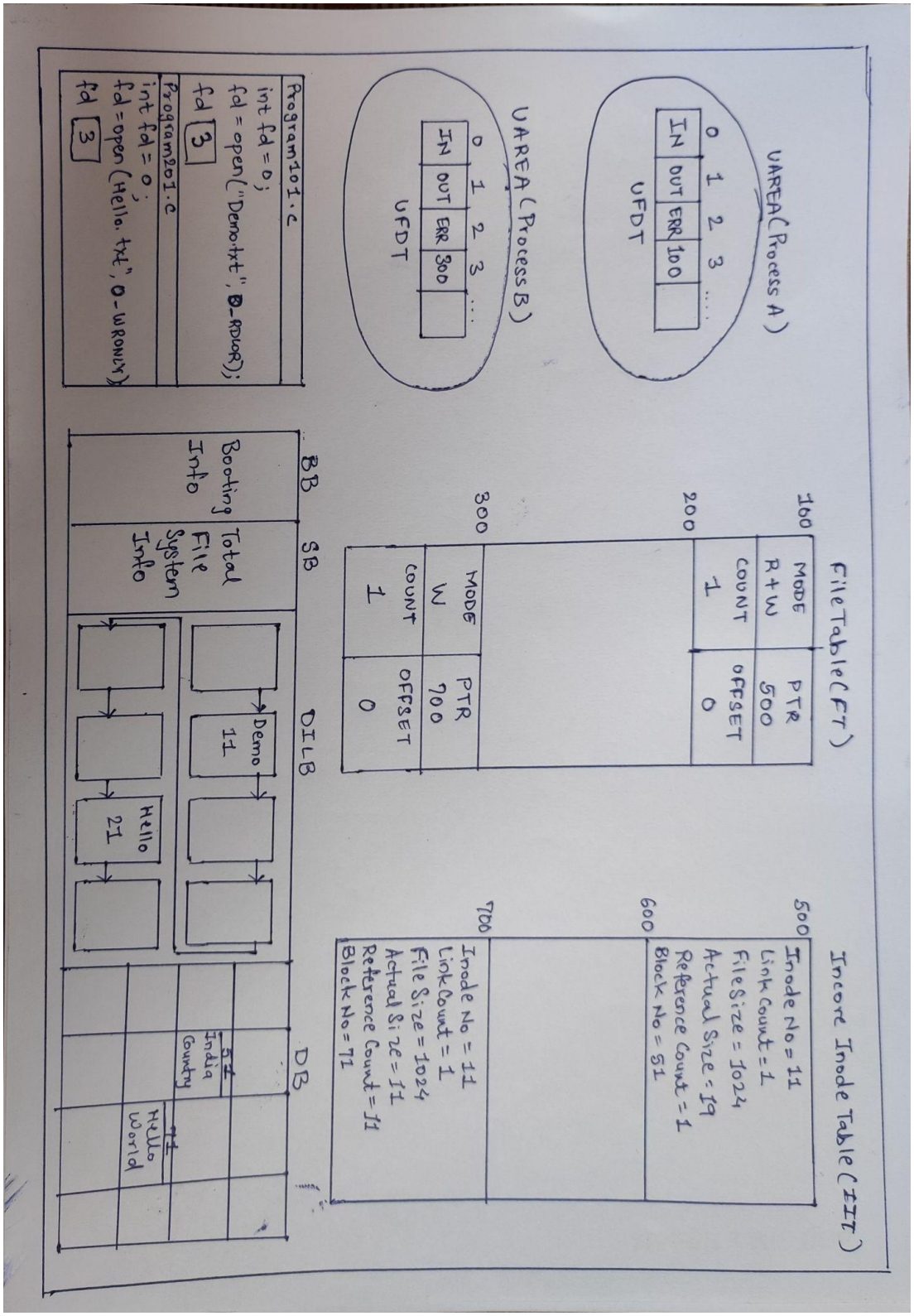
FILETABLE -



UFDT -



Complete File System Diagram



Code of the project

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<iostream>
#include<io.h>

#define MAXINODE 5

#define READ 1
#define WRITE 2

#define MAXFILESIZE 1024

#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    2   3
```

```
    struct inode *next;
}INODE, *PINODE, **PPONODE;

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[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 : Used to read 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 you
want to write.\n");
```

```
    }
    else if(strcmp(name,"ls") == 0)
    {
        printf("Description : Used to list all information of files.\n");
        printf("Usage : ls\n");
    }
    else if(strcmp(name,"stat") == 0)
    {
        printf("Description : Used to display all information of
files.\n");
        printf("Usage : stat File_name.\n");
    }
    else if(strcmp(name,"fstat") == 0)
    {
        printf("Description : Used to display the information of
files.\n");
        printf("Usage : stat File_Descriptor.\n");
    }
    else if(strcmp(name,"truncate") == 0)
    {
        printf("Description : Used to remove the data from 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 the opened file.\n");
        printf("Usage : close File_name.\n");
    }
    else if(strcmp(name,"closeall") == 0)
    {
        printf("Description : Used to close all the opened files.\n");
        printf("Usage : closeall.\n");
    }
    else if(strcmp(name,"lseek") == 0)
    {
```

```
        printf("Description : Used to change the 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");
    }
}

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 the opened files\n");
    printf("read : To Read the contents from th file\n");
    printf("write : To Write the content from the file\n");
    printf("exit : To Terminate file system\n");
    printf("stat : To Display information of file using name\n");
    printf("fstat : To Display information of file using file
descriptor\n");
    printf("truncate : To Remove all the data from the file]\n");
    printf("rm : To Delete 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 = 0;
    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.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) + (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)
        {
            if(UFDTArr[i].ptrfiletable -> readoffset = 0);
            if(UFDTArr[i].ptrfiletable -> writeoffset = 0);
            (UFDTArr[i].ptrfiletable -> ptrinode -> ReferenceCount)--;
            break;
        }
    }
    i++;
}

int LseekFile(int fd, int iSize, 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) + iSize) >
UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize)    return -1;
            if(((UFDTArr[fd].ptrfiletable -> readoffset) + iSize) < 0)
return -1;
            (UFDTArr[fd].ptrfiletable -> readoffset) =
(UFDTArr[fd].ptrfiletable -> readoffset) + iSize;
        }
        else if(from == START)
        {
```

```
            if(iSize > (UFDTArr[fd].ptrfiletable -> ptrinode ->
FileActualSize))    return -1;
            if(iSize < 0)    return -1;
            (UFDTArr[fd].ptrfiletable -> readoffset) = iSize;
        }
        else if(from == END)
        {
            if((UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) +
iSize > MAXFILESIZE)  return -1;
            if(((UFDTArr[fd].ptrfiletable -> readoffset) + iSize) < 0)
return -1;
            (UFDTArr[fd].ptrfiletable -> readoffset) =
(UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) + iSize;
        }
    }
    else if(UFDTArr[fd].ptrfiletable -> mode == WRITE)
    {
        if(from == CURRENT)
        {
            if(((UFDTArr[fd].ptrfiletable -> writeoffset) + iSize) >
MAXFILESIZE)   return -1;
            if(((UFDTArr[fd].ptrfiletable -> writeoffset) + iSize) < 0)
return -1;
            if(((UFDTArr[fd].ptrfiletable -> writeoffset) + iSize) >
(UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize))
            ((UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) =
(UFDTArr[fd].ptrfiletable -> writeoffset) + iSize);
            (UFDTArr[fd].ptrfiletable -> writeoffset) =
(UFDTArr[fd].ptrfiletable -> writeoffset) + iSize;
        }
        else if(from == START)
        {
            if(iSize > MAXFILESIZE)     return -1;
            if(iSize < 0)   return -1;
            if(iSize > (UFDTArr[fd].ptrfiletable -> ptrinode ->
FileActualSize))
            (UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) =
iSize;
            (UFDTArr[fd].ptrfiletable -> writeoffset) = iSize;
        }
```

```
        else if(from == END)
        {
            if((UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) +
iSize > MAXFILESIZE)  return -1;
            if(((UFDTArr[fd].ptrfiletable -> writeoffset) + iSize) < 0)
return -1;
            (UFDTArr[fd].ptrfiletable -> writeoffset) =
(UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize) + iSize;
        }
    }
}

void ls_file()
{
    int i = 0;
    PINODE temp = head;

    if(SUPERBLOCKobj.FreeInodes == 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-----------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 Only\n");
    else if(temp -> permission == 3)
        printf("File Permission : Read and Write\n");

printf("-----------------------------------------------------------------\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 Only\n");
    else if(temp -> permission == 3)
        printf("File Permission : Read and Write\n");

printf("-----------------------------------------------------------------\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("Marvellous VFS : > ");

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

        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 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 Marvellous 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]",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 file
descriptor : %d\n",ret);
                if(ret == -1)
                    printf("ERROR : Incorrect parmeters\n");
                if(ret == -2)
                    printf("ERROR : There are 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 file
descriptor : %d\n",ret);
                if(ret == -1)
                    printf("ERROR : Incorrect parmeters\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 : File empty\n");
    if(ret == 0)
        printf("ERROR : It is not regular file\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;
}
```

Screenshots of the output

help - to guide the user about the calls

```
C:\WINDOWS\system32\cmd.

C:\Users\1951a\LB\CVFS>myexe
DILB created successfully.
Marvellous VFS : > help
ls : To List out all files
clear : To Clear console
open : To Open the file
close : To Close the file
closeall : To Close all the opened files
read : To Read the contents from th file
write : To Write the content from the file
exit : To Terminate file system
stat : To Display information of file using name
fstat : To Display information of file using file descriptor
truncate : To Remove all the data from the file]
rm : To Delete the file
Marvellous VFS : >
```

create - to create a file

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > create demo.txt 3
Marvellous VFS : > ls

File Name       Inode number    File size       Link Count
------------------------------------------------------------
demo.txt                0               0               1
------------------------------------------------------------
Marvellous VFS : > S
```

write - to write the data in the file

```
Marvellous VFS : > write demo.txt
Enter the data :
This is a demo file for testing the application.
Marvellous VFS : >
```

read - to read the data from the file

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > read demo.txt 11
This is a dMarvellous VFS : >
```

ls - to list out all files

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > ls

File Name       Inode number    File size       Link Count
------------------------------------------------------------
demo.txt                0               48              1
------------------------------------------------------------
Marvellous VFS : >
```

stat - To Display information of file using name

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > stat demo.txt

-----------Statistical Information about the file-----------
File Name : demo.txt
Inode Number : 0
File Size : 1024
Actual File Size : 48
Link Count : 1
Reference Count : 1
File Permission : Read and Write
-------------------------------------------------------------
Marvellous VFS : >
```

open - to open the file

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > open demo.txt 3
File is successfully opened with file descriptor : 1
Marvellous VFS : >
```

closeall - to close all the opened files

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > closeall
All files closed successfully
Marvellous VFS : >
```

lseek (before)

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > read demo.txt 48
This is a demo file for testing the application.Marvellous VFS : >
```

lseek (after)

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > lseek demo.txt 2 1
Marvellous VFS : > read demo.txt 48
is is a demo file for testing the application."Marvellous VFS : >
```

truncate - to remove all the data from the file

```
C:\WINDOWS\system32\cmd.
Marvellous VFS : > truncate demo.txt
Marvellous VFS : > read demo.txt 48
ERROR : Reached at the end of the file
Marvellous VFS : >
```

rm - to delete the file

```
C:\WINDOWS\system32\cmd.

Marvellous VFS : > create demmo.txt 3
Marvellous VFS : > ls

File Name       Inode number    File size       Link Count
------------------------------------------------------------
demmo.txt               0               0               1
------------------------------------------------------------
Marvellous VFS : > rm demmo.txt
Marvellous VFS : > ls

File Name       Inode number    File size       Link Count
------------------------------------------------------------
------------------------------------------------------------
Marvellous VFS : >
```