Documentation of Customised Virtual File System:

Name:

Customised Virtual File system:

TechnologyUsed:

System Programming using C.

User Interface Used:

Command User Interface(CLI)

Platform Required:

Windows NT platform OR Linux Distributions

Hardware Requirements:

Operating System:Windows 7 or above.

CPU:core i3 and above.

Ram:Minimum 4GB

Description of the Project:

In this Project we emulate all data structrures which are used by operating system to manage File system oriented tasks.

As the name suggest its virtual because we maintain all records in Primary storage.

In this project we create all data structures which are required for Filesystems as Inode Inode Table,File Table,UAREA,User Area Descriptor Table,Super Block,Disk Inode List Block,Data Block,Boot block etc.

We have used the inbuilt system calls. In this project we have implemented all the systems calls such as open,Close Read,Write,Create,RM,LS,stat,fstat etc.

For the implementation of above system call we have created our own data structres.

By using this cvfs project we can get overview of Unix file system on any platorm.

Data Structures Used In the Project:

DILB Linkedlist:

Here in this Project we have used Singly Linear LinkedList.By using the singly linear linkedlist we have created the file.As the No of Files creation is realted to inode.Hence we have created the linkedlist of Inodes.Inode contains the information about the single file.

UFDT Array:

UFDT Array is array which stores the address of file tables.

Diagram of data Structures used in the project:

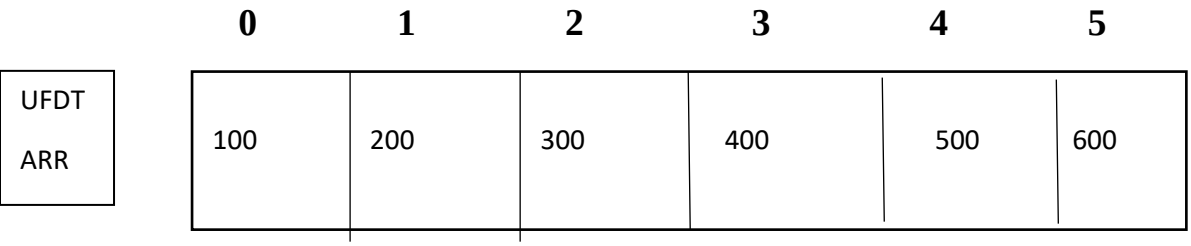
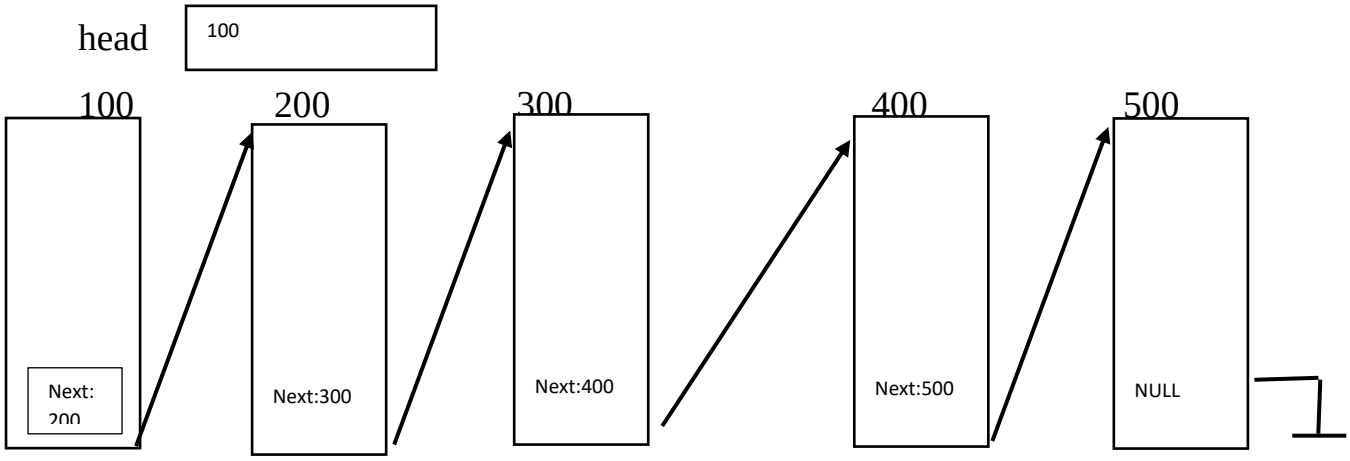


Diagram of Linked List:



The Flow of the Project:

The working flow of the project starts from the Superblock,because it conatins the Total inodes and free inode,As inode is used to create the File.

Firstly the UFDT Array is there which stores the address of file table.After reaching to the file table we get Information about file As the File Read offset,write offset,Count,Mode of the file,Ptrinode.This is the pointer to the inode,which points to that specfic inode which addres is stored in it.

After that in the Inode there are certain paramters like the Name of the file,Inode Number,Actual size or data it contains in bytes,Type of the file,Buffer contains the address to the buffer,Linkcount,reference count,Permission to the file.and next pointer which stores the address of the next inodes.

And Buffer ponits to the data block where in we get the actual data in the file.

There are various functions as Help and Man which provides us the inofrmation.By using which User gets the documentaion of the project.

Source Code:

```
#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 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;
    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:used to read new data from regular file\n");
 printf("Usage:create File_name  No_of_Bytes_To_Read\n");

}
else if(strcmp(name,"write")==0)
{
 printf("Description:used to write new data into regular file\n");
 printf("Usage:write File_name  \nAfter this enter the data that we want to write");

}
else if(strcmp(name,"Is")==0)
{
 printf("Description:used to list all information of files \n");
 printf("Usage:Is\n");

}
else if(strcmp(name,"stat")==0)
{
 printf("Description:used to Display all information of files \n");
 printf("Usage:Is\n");

}
else if(strcmp(name,"fstat")==0)
{
 printf("Description:used to Display all information of files \n");
 printf("Usage:stat File_Descriptor\n");

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

```
{
 printf("Description:used to remove data from files \n");
 printf("Usage:truncate File_name\n");

}
else if(strcmp(name,"open")==0)
{
 printf("Description:used to open  \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 information of files \n");
 printf("Usage:close File_name\n");

}
else if(strcmp(name,"Iseek")==0)
{
 printf("Description:used to change file offset \n");
 printf("Usage:Iseek 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 etry available");

    }

}
void DisplayHelp()
{
    printf("Is: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 into file\n");
    printf("write:To write contents into the file\n");
    printf("exit:To terminate file system\n");
    printf("stat:To Display information of the file using name\n");
    printf("fstat:To Display information of file using descriptor\n");
    printf("truncate:To Remove all 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=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 successully\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[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;
}
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);
    }
    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)==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)--;
        }
        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;  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))
```

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

        }

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

    if (SUPERBLOCKobj.FreeInode == MAXINODE)
    {
        printf("Error :There are no such 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 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\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 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\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",command[0],command[1],command[2],command[3]);
          if (count==1)
          {
              if (strcmp(command[0],"Is")==0)
```

```
    {
        Is_file();
    }
    else if (strcmp(command[0],"closeAll")==0)
    {
        CloseAllFile();
        printf("All files closed successfully");
    }
    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");
      break;
    }
    else
    {
      printf("Error:");
      continue;
    }

 }
else if(count==2)
{
```

```
 if (strcmp(command[0],"stat")==0)
 {
   ret=stat_file(command[1]);
   if(ret==-1)
        printf("ERROR:Incorrect parameters");
   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");
   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:Incorrect parameters\n");
   if(ret==-2)
        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:Incorrect parameters\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 parameters\n");
      continue;
    }
    printf("Enter the data");
    scanf("%[^\n]",arr);

    ret=strlen(arr);
    if (ret==0)
    {
      printf("Error :Incorrect parameters\n");
      continue;
    }
    ret=WriteFile(fd,arr,ret);
    if (ret==-1)
    {
      printf("Error :Permission denied");
    }
    if (ret==-2)
    {
      printf("Error :There is no sufficient memory to write");
    }
    if (ret==-3)
    {
      printf("Error :It is not regular file");
```

```
    }
}
 else if(strcmp(command[0],"truncate")==0)
 {
        ret=truncate_File(command[1]);
        if (ret==-1)
           printf("Error :Incorrect parameter\n");

 }
 else
 {
   printf("ERROR:command not found!!!");
   continue;
 }
}
else if (count==3)
{
   if (strcmp(command[0],"create")==0)
   {
       ret=CreateFile(command[1],atoi(command[2]));

       if (ret >= 0)
              printf("File is successsfully creadted with descriptot %d\n",ret);

        if (ret==-1)
             printf("ERROR:incorrect pararmetres",ret);

        if (ret==-2)
             printf("ERROR:There is no inodes");
         if (ret==-3)
             printf("ERROR:File already exists");
         if (ret==-4)
             printf("ERROR:Memory allocation failure");
         continue;
```

```
}
else if (strcmp(command[0],"open")==0)
{
    ret=OpenFile(command[1],atoi(command[2]));
    if (ret>=0)
        printf("File is successsfully creadted with descriptot %d\n",ret);
     if (ret==-1)

        printf("ERROR:incorrect pararmetres",ret);
      if (ret==-2)

        printf("ERROR:File noy exists");
      if (ret==-3)

        printf("ERROR:Permission denied");
      continue;
}

else if (strcmp(command[0],"read")==0)
{
    fd=GetFDFromName(command[1]);
    if (fd==-1)
    {
        printf("Error Incorrect parameters\n");
        continue;
    }
    ptr = (char*)malloc(sizeof(atoi(command[2]))+1);
    if (ptr==NULL)
    {
        printf("Error :Memory Allocation fialure");
        continue;
    }
    ret=ReadFile(fd,ptr,atoi(command[2]));
    if (ret==-1)
```

```
            printf("ERROR:File not existing\n");
        if (ret==-2)
            printf("ERROR:Permission denied");
        if (ret==-3)
            printf("ERROR:Reached at end of 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("Error:command not found");
      continue;
   }

}
else if (count==4)
{
   if (strcmp(command[0],"Iseek")==0)
   {
      fd=GetFDFromName(command[1]);
      if (fd==-1)
      {
          printf("Error:incorrect parrameter");
          continue;
      }
```

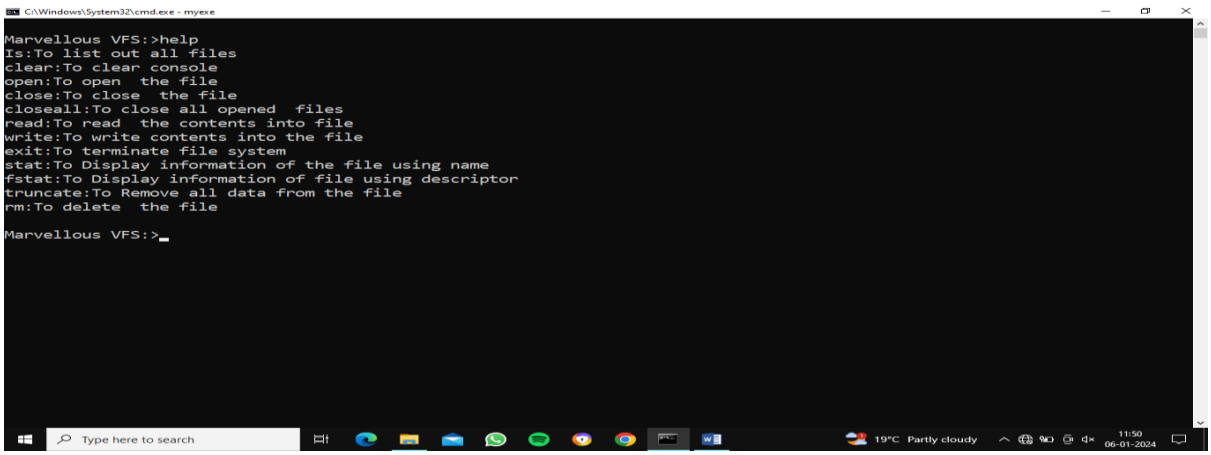
```
                ret=LseekFile(fd,atoi(command[2]),atoi(command[3]));
                if (ret==-1)
                {
                    printf("Error :unable to perform Iseek");


                }
            }
            else
            {
                printf("command not found");
                continue;
            }

        }
        else
        {
            printf("command not found");
            continue;
        }
    }
    return 0;
}
```

Screen Shots of Output:

1)help Command:



2)DILB:

```
C:\Windows\System32\cmd.exe - myexe
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

G:\LB\CVFS>g++ cvfs.cpp -o myexe

G:\LB\CVFS>myexe
DILB created successully

Marvellous VFS:>
```

3)Man Command:

```
C:\Windows\System32\cmd.exe - myexe
Usage:create File_name Permission

Marvellous VFS:>man write
Description:used to write new data into regular file
Usage:write File_name
After this enter the data that we want to write
Marvellous VFS:>man read
Description:used to read new data from regular file
Usage:create File_name  No_of_Bytes_To_Read

Marvellous VFS:>man fstat
Description:used to Display all information of files
Usage:stat File_Descriptor

Marvellous VFS:>man stat
Description:used to Display all information of files
Usage:Is

Marvellous VFS:>man rm
Description:used to delete the file
Usage:rm File_Name

Marvellous VFS:>man ls
Error :No manual etry available
Marvellous VFS:>man Is
Description:used to list all information of files
Usage:Is

Marvellous VFS:>
```

4)Create ,READ WRITE,LS:

```
C:\Windows\System32\cmd.exe - myexe

Marvellous VFS:>create demo1.txt 3
File is successsfully creadted with descriptot 0

Marvellous VFS:>write demo1.txt
Enter the data

Marvellous VFS:>write demo1.txt
Enter the data hello cvfs

Marvellous VFS:>read demo1.txt
ERROR:command not found!!!
Marvellous VFS:>read demo1.txt 5
♬ hel
Marvellous VFS:>read demo1.txt 4
lo c
Marvellous VFS:>read demo1.txt 3
vfs
Marvellous VFS:>Is

File Name       Inode number    File size       Link count
------------------------------------------------------------------------
demo1.txt               1               12              1
------------------------------------------------------------------------

Marvellous VFS:>
```

5)FSTAT AND STAT:

```
C:\Windows\System32\cmd.exe - myexe
------------------------------------------------------------------------

Marvellous VFS:>fstat demo1.txt

------------Statistical Information about file---------
File name:demo1.txt
Inode Number 1
File size :2048
Actual File Size :12
Link count :1
Reference count :1
File permission : Read and Write
-----------------------------------------------


Marvellous VFS:>stat demo1.txt

------------Statistical Information about file-----------------
File name:demo1.txt
Inode Number 1
File size :2048
Actual File Size :12
Link count :1
Reference count :1
File permission : Read and Write
-----------------------------------------------------------------


Marvellous VFS:>
```

6)RM:

```
C:\Windows\System32\cmd.exe - myexe
File name:demo1.txt
Inode Number 1
File size :2048
Actual File Size :12
Link count :1
Reference count :1
File permission : Read and Write
-----------------------------------------------


Marvellous VFS:>stat demo1.txt

------------Statistical Information about file-----------------
File name:demo1.txt
Inode Number 1
File size :2048
Actual File Size :12
Link count :1
Reference count :1
File permission : Read and Write
---------------------------------------------------------------


Marvellous VFS:>rm demo1.txt

Marvellous VFS:>Is
Error :There are no such files

Marvellous VFS:>
```