#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<io.h>

#define MAXINODE 50

#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 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[MAXINODE];

SUPERBLOCK SUPERBLOCKobj; PINODE head=NULL;

void man(char \*name)

{

if(name==NULL)

{

return;

}

if(\_stricmp(name,"create")==0)

{

printf("Description : Used to create new regular file \n"); printf("Usage : create File\_name Permission \n");

}

else if(\_stricmp(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(\_stricmp(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(\_stricmp(name,"ls")==0)

{

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

printf("Usage : ls \n");

}

else if(\_stricmp(name,"stat")==0)

{

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

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

}

else if(\_stricmp(name,"fstat")==0)

{

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

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

}

else if(\_stricmp(name,"truncate")==0)

{

printf("Description : Used to remove data from file \n"); printf("Usage : truncate File\_name \n");

}

else if(\_stricmp(name,"open")==0)

{

printf("Description : Used to open existing file \n"); printf("Usage : open File\_name mode \n");

}

else if(\_stricmp(name,"close")==0)

{

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

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

}

else if(\_stricmp(name,"closeall")==0)

{

printf("Description : Used to close all opened file \n"); printf("Usage : closeall \n"); }

else if(\_stricmp(name,"lseek")==0)

{

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

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

}

else if(\_stricmp(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 file \n"); printf("read : To read the contents from file \n"); printf("write : To write contents into 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 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)

{

if(\_stricmp((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=newn->ReferenceCount=0;

newn->FileType=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;

}

if(Get\_Inode(name)!=NULL)

{

return -3;

}

(SUPERBLOCKobj.FreeInode)--;

while(temp!=NULL)

{

if(temp->FileType==0)

{

break;

}

temp-temp->next;

}

while(i<MAXINODE)

{

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

{

break;

}

i++; }

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

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

{

return -4;

}

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

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 < MAXINODE)

{

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 < MAXINODE)

{

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))

(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;

}

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

printf("------------------------------------------------------");

while(temp != NULL)

{

if((temp->FileType) != 0)

{

printf("\n%s\t\t%d\t\t%d\t\t%d\n",temp->FileName,temp-

>InodeNumber,temp->FileActualSize,temp->LinkCount);

}

temp = temp->next;

}

printf("---------------------------------------------------------"); }

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\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 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[1024];

InitialiseSuperBlock();

CreateDILB();

while(1)

{

fflush(stdin); strcpy\_s(str,""); printf("\n Marvellous VFS : >");

fgets(str,80,stdin);

count = sscanf(str,"%s %s %s

%s",command[0],command[1],command[2],command[3]);

if(count == 1)

{

if(\_stricmp(command[0],"ls") == 0)

{

ls\_file();

}

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

{

CloseAllFile();

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

continue;

}

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

{

system("cls");

continue;

}

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

{

DisplayHelp(); continue;

}

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

{

printf("Terminating the Marvellous Virtual File System \n"); break;

}

else {

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

continue;

}

}

else if(count == 2)

{

if(\_stricmp(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(\_stricmp(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(\_stricmp(command[0],"close") == 0)

{

ret = CloseFileByName(command[1]);

if(ret == -1)

{

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

}

continue;

}

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

{

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

if(ret == -1)

{

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

}

continue; }

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

{

man(command[1]);

}

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

{

fd = GetFDFromName(command[1]);

if(fd == -1)

{

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

continue;

}

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

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 \n");

}

if(ret == -2)

{

printf("ERROR : There is no sufficient memory to write

\n");

}

if(ret == -3)

{

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

}

}

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

{

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

if(ret == -1)

{

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

}

}

else

{

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

}

} else if(count == 3)

{

if(\_stricmp(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 Parameters \n");

}

if(ret == -2)

{

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

}

if(ret == -3)

{

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

}

if(ret ==-4)

{

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

}

continue;

}

else if(\_stricmp(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 Parameters \n");

}

if(ret == -2)

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

}

if(ret == -3)

{

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

}

continue;

}

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

{ fd = GetFDFromName(command[1]);

if(ret == -1)

{

printf("ERROR : Incorrect Parameters \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 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("\n ERROR : Command not found !!! \n"); continue;

}

}

else if(count == 4)

{

if(\_stricmp(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("\n ERROR : Command not found !!! \n");

continue;

} }

else

{

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

continue;

}

}

return 0;

}