//Header.h

#include <stdio.h>

#include <string>

#include <io.h>

#include <stdlib.h>

#define MAXINODE 50 //set maximum number of inode

#define MAXFILESIZE 1024 //set maximum size of file ( don't change )

#define MAX\_UFDT 50 //set maximum number of file that one process can open

//MODE OF FILE

#define READ 1

#define WRITE 2

//FILE TYPE

#define REGULAR 1 //only regular is use

#define SPECIAL 2

//FOR LSEEK FUNCTION

#define START 0

#define CURRENT 1

#define END 2

//SUCCESS

#define SUCCESS 0

//ERROR CODE

#define ERR\_INCORRECT\_PARMETER -1

#define ERR\_INODE\_FULL -2

#define ERR\_FILE\_EXISTS -3

#define ERR\_MEMORY\_ALLOCATION\_FAIL -4

#define ERR\_UFDT\_FULL -5

#define ERR\_FILE\_NOT\_FOUND -6

#define ERR\_FILE\_IS\_NOT\_CLOSE -7

#define ERR\_PERMISSION\_DENIED -8

#define ERR\_IN\_SUFFICIENT\_MEMORY -9

#define ERR\_REACH\_AT\_END -10

#define ERR\_FILE\_ALREADY\_OPEN -11

#define ERR\_UNABLE\_TO\_PERFORM\_LSEEK -12

typedef struct superblock

{

int TotalInode;

int FreeInode;

}SUPERBLOCK , \*PSUPERBLOCK;

/\* Here we consider dynamic Inode and static Inode as one because we creating Filesystem on RAM.

But it will more similar to static incode ie Disk Inode List Block \*/

typedef struct inode

{

char FileName[50]; /\*In actual inode struct , file name is not store. For simplicity we add this field.\*/

int InodeNumber; /\*This inode struct is on RAM (dynamic inode) so we store Inodenumber value inside struct.

This field is absent in static inode (Inode in Disk Incode List Table ). \*/

int FileSize; /\*Total memory(in byte) allocated for file ( block addition) \*/

int FileActualSize; /\*Total memory(in byte) use by file within allocated memory (byte additon) \*/

int FileType; /\*Macro is define ( REGULAR , SPECIAL \*/

char \*Buffer; /\*In actual Inode there is Array of 15 pointer which point to bock in data block.

For Simplicity we as use char pointer. Afterward we will allocated memory dynamically \*/

int LinkCount; /\*LinkCount store number of hard link to this Inode (ie. number of name given to inode )

For Simplicity we consider LinkCount = 1 \*/

int ReferenceCount; /\*ReferenceCount tells how many process are using this Inode \*/

int Permission; /\*Macro is define:- READ , WRITE , or both ( READ + WRITE ) \*/

struct inode \*next;

}INODE;

typedef INODE \* PINODE;

typedef struct fileTable

{

int readOffset; /\* In actual sytem there is one offset , but for simplcity we have use two offset\*/

int writeOffset;

int count;

int mode;

PINODE ptrinode;

}FILETABLE , \*PFILETABLE;

typedef struct ufdt //USER FILE DESCRIPTION TABLE

{

PFILETABLE ptrfiletable;

}UFDT;

//GOBAL VARIABLE DECALARATION

extern UFDT UFDTArr[MAX\_UFDT]; //One Process can open MAX\_UFDT

extern SUPERBLOCK SUPERBOLCKobj;

extern PINODE head;

//FUNCTION DEFINTION

void DisplayHelp();

void man(char \*);

void DisplayError(int );

void InitialiseSuperBlock();

void CreateDILB();

void ls\_file();

int CreateFile( char\* , int );

PINODE GetInode(char\*);

int CloseFileByName(char \*);

void CloseAllFile();

int rm\_File( char \* );

int GetFDFromName(char \*);

int StatFile( char \* );

int FstatFile( int );

void DisplayStatInfoOfInode(PINODE);

int OpenFile( char\* , int );

int WriteFile(int , char \* , int );

int ReadFile( int , char \* , int );

int LseekFile( int , int , int );

int truncateFile( char \* );

//InfoDisplay.cpp

#include "Header.h"

extern UFDT UFDTArr[MAX\_UFDT]; //One Process can open MAX\_UFDT

extern SUPERBLOCK SUPERBOLCKobj;

extern PINODE head;

void DisplayHelp()

{

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

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

printf("create : To Create the new file\n");

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

}

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");

printf(" Permission can be 1 2 3");

}

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

{

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

printf("Usage : read File\_name No\_of\_byte\_to\_Read\n");

}

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

{

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

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

printf(" 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 : fstat 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 Change\_In\_Offset StarPoint\n");

printf(" StarPoint can be START 0 CURRENT 1 END 2\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.");

}

}

void DisplayError( int errorCode )

{

switch(errorCode)

{

case ERR\_INCORRECT\_PARMETER :

printf("ERROR: InCorrect Paramter\n");

break;

case ERR\_INODE\_FULL :

printf("ERROR: InNode is Full\n");

break;

case ERR\_FILE\_EXISTS :

printf("ERROR: File is Already exist\n");

break;

case ERR\_MEMORY\_ALLOCATION\_FAIL:

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

break;

case ERR\_UFDT\_FULL :

printf("ERROR: Process UFDT is Full\n");

break;

case ERR\_FILE\_NOT\_FOUND :

printf("ERROR: File no found\n");

break;

case ERR\_FILE\_IS\_NOT\_CLOSE :

printf("ERROR: File is not close\n");

break;

case ERR\_PERMISSION\_DENIED :

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

break;

case ERR\_IN\_SUFFICIENT\_MEMORY :

printf("ERROR: Insufficients Memory\n");

break;

case ERR\_REACH\_AT\_END :

printf("ERROR: file is reach at end\n");

break;

case ERR\_FILE\_ALREADY\_OPEN :

printf("ERROR: File is already open\n");

break;

case ERR\_UNABLE\_TO\_PERFORM\_LSEEK :

printf("ERROR: Unacle to perform lseef\n");

break;

}

}

//Helper.cpp

#include "Header.h"

UFDT UFDTArr[MAX\_UFDT]; //One Process can open MAX\_UFDT

SUPERBLOCK SUPERBOLCKobj;

PINODE head = NULL;

void InitialiseSuperBlock()

{

int i = 0 ;

SUPERBOLCKobj.TotalInode = MAXINODE;

SUPERBOLCKobj.FreeInode = MAXINODE;

while ( i < MAX\_UFDT)

{

UFDTArr[i].ptrfiletable = NULL;

i++;

}

}

void CreateDILB()

{

PINODE newInode = NULL;

PINODE temp = head;

int i = 1; /\* Incode number 0 is not use. Because after deleting a file from directory . In that directory beside that file name incode value is keep 0. To indicate that the file is been delete\*/

while( i < MAXINODE )

{

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

strcpy\_s(newInode->FileName,"");

newInode -> InodeNumber = i;

newInode -> LinkCount = newInode->ReferenceCount = 0;;

newInode -> Buffer = NULL;

newInode -> next = NULL;

newInode -> FileSize = 0;

newInode -> FileType = 0;

if(temp == NULL)

{

head = newInode;

temp = head;

}

else

{

temp->next = newInode;

temp = newInode;

}

i++;

}

}

//open and create are almost similar

int OpenFile( char \*name , int mode)

{

int i = 0;

PINODE temp = NULL;

//mode can't be more than 3 or less than 1

if( name == NULL || mode <= 0 || mode >= 4 )

return ERR\_INCORRECT\_PARMETER;

//Checking is file exist or not

temp = GetInode(name);

if(temp == NULL)

return ERR\_FILE\_NOT\_FOUND;

//Filter - not to open file if its already open

i = GetFDFromName(name);

if( i != ERR\_FILE\_NOT\_FOUND )

return ERR\_FILE\_ALREADY\_OPEN;

if( temp->Permission < mode )

return ERR\_PERMISSION\_DENIED;

while( i < MAX\_UFDT)

{

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

break;

i++;

}

if( i == MAX\_UFDT )

return ERR\_UFDT\_FULL;

/\*I Initialise ptablefile

open Inode at i position\*/

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

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

return ERR\_MEMORY\_ALLOCATION\_FAIL;

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

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

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

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

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

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

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

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

return i;

}

void ls\_file()

{

PINODE temp = head;

if(SUPERBOLCKobj.FreeInode == MAXINODE )

{

printf("There is no files\n");

return;

}

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

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

while( temp != NULL )

{

if( temp -> LinkCount != 0 )

{

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

}

temp = temp->next;

}

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

}

int CreateFile( char \*name , int permission )

{

int i = 0;

PINODE temp = head;

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

return ERR\_INCORRECT\_PARMETER;

if( SUPERBOLCKobj.FreeInode == 0 )

return ERR\_INODE\_FULL;

if( GetInode(name) != NULL )

return ERR\_FILE\_EXISTS;

/\* new logic \*/

while(temp->LinkCount != 0 )

{

temp = temp->next;

}

//Here we get free Inode Pointer(temp)

while( i < MAX\_UFDT )

{

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

break;

i++;

}

if( i == MAX\_UFDT)

return ERR\_UFDT\_FULL;

//Here we get Free index of Free UFDT

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

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

return ERR\_MEMORY\_ALLOCATION\_FAIL;

UFDTArr[i].ptrfiletable->count = 1; //for simplicity we give 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->Permission = permission;

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

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

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

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

(SUPERBOLCKobj.FreeInode)--;

return i;

}

PINODE GetInode( char\* name)

{

PINODE temp = head;

if( name == NULL )

return NULL;

while( temp != NULL )

{

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

break;

temp = temp->next;

}

return temp;

}

int CloseFileByName(char \* name )

{

int i = 0;

i = GetFDFromName(name);

if( i == ERR\_FILE\_NOT\_FOUND )

return i;

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

free(UFDTArr[i].ptrfiletable);

UFDTArr[i].ptrfiletable = NULL;

return SUCCESS;

}

void CloseAllFile()

{

int i = 0;

while( i < MAX\_UFDT )

{

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

{

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

free(UFDTArr[i].ptrfiletable);

UFDTArr[i].ptrfiletable = NULL;

}

i++;

}

}

int rm\_File( char \*name)

{

int fd = 0;

PINODE temp = head;

if( name == NULL )

return ERR\_INCORRECT\_PARMETER;

fd = GetFDFromName(name);

if(fd >= 0)

return ERR\_FILE\_IS\_NOT\_CLOSE;

// File cannot be close if its is open

while( temp != NULL )

{

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

break;

temp = temp->next;

}

if( temp == NULL )

return ERR\_FILE\_NOT\_FOUND;

(temp->LinkCount)--;

//Remove Inode Data if linkCount == 0

if( temp->LinkCount == 0 )

{

free(temp->Buffer);

temp->FileActualSize = 0;

strcpy\_s(temp->FileName,"");

(SUPERBOLCKobj.FreeInode)++;

}

return SUCCESS;

}

int GetFDFromName( char\* name)

{

int i = 0;

while( i < MAX\_UFDT)

{

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

{

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

break;

}

i++;

}

if( i == MAX\_UFDT)

return ERR\_FILE\_NOT\_FOUND;

return i;

}

int StatFile( char \* name )

{

PINODE temp = head;

if( name == NULL )

return ERR\_INCORRECT\_PARMETER;

while( temp != NULL )

{

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

break;

temp = temp-> next;

}

if( temp == NULL )

return ERR\_FILE\_NOT\_FOUND;

DisplayStatInfoOfInode( temp );

return SUCCESS;

}

int FstatFile(int fd)

{

PINODE temp = NULL;

if( fd < 0 || fd > MAX\_UFDT )

return ERR\_INCORRECT\_PARMETER;

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

return ERR\_FILE\_NOT\_FOUND;

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

DisplayStatInfoOfInode( temp );

return SUCCESS;

}

void DisplayStatInfoOfInode(PINODE temp)

{

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);

printf("File Type : Regular file\n");

if(temp->Permission == 1 )

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

else if(temp->Permission == 2 )

printf("File Permission : Write\n");

else

printf("File Permission :Read and Write\n");

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

}

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

{

//checking permission given at open file

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

return ERR\_PERMISSION\_DENIED;

if( ( UFDTArr[fd].ptrfiletable->writeOffset + isize ) >= MAXFILESIZE)

return ERR\_IN\_SUFFICIENT\_MEMORY;

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

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

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

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

return SUCCESS;

}

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

{

int read\_size;

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

return ERR\_PERMISSION\_DENIED;

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

return ERR\_REACH\_AT\_END;

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 += read\_size;

return read\_size;

}

else

{

strncpy(arr ,

( UFDTArr[fd].ptrfiletable->ptrinode->Buffer) +

(UFDTArr[fd].ptrfiletable->readOffset) , isize);

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

}

return isize;

}

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

{

if( ( fd < 0 ) || ( fd > MAX\_UFDT ) )

{

return ERR\_INCORRECT\_PARMETER;

}

if( from > 2 || from < 0)

{

return ERR\_INCORRECT\_PARMETER;

}

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

return ERR\_FILE\_NOT\_FOUND;

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

{

if( from == CURRENT )

{

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

}

else if( from == START )

{

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

if( size < 0 )

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

}

else if( from == END)

{

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

}

else if( from == START )

{

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

if( size < 0 )

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

}

else if( from == END)

{

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

return ERR\_UNABLE\_TO\_PERFORM\_LSEEK;

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

}

}

return SUCCESS;

}

int truncateFile( char \* name )

{

int fd = GetFDFromName(name);

if( fd == ERR\_FILE\_NOT\_FOUND )

return ERR\_FILE\_NOT\_FOUND;

memset( UFDTArr[fd].ptrfiletable->ptrinode->Buffer , 0 , 1025 );

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

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

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

return SUCCESS;

}

//Main.cpp

#include "Header.h"

extern UFDT UFDTArr[MAX\_UFDT]; //One Process can open MAX\_UFDT

extern SUPERBLOCK SUPERBOLCKobj;

extern PINODE head;

int main()

{

char command[4][80];

char str[80];

int count;

int ret;

int fd ;

char arr[1024];

char \*ptr;

InitialiseSuperBlock();

CreateDILB();

while(1)

{

fflush(stdin);

strcpy\_s(str,"");

printf("\nMarvellous VFS: > ");

fgets(str,80,stdin);

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

if(count == 1)

{

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

{

DisplayHelp();

continue;

}

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

{

system("cls");

continue;

}

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

{

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

break;

}

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

{

ls\_file();

continue;

}

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

{

CloseAllFile();

continue;

}

else

{

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

continue;

}

}

else if( count == 2 )

{

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

{

man(command[1]);

continue;

}

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

{

ret = CloseFileByName( command[1] );

if( ret == SUCCESS )

printf("Files is successfully Close\n");

else

DisplayError(ret);

continue;

}

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

{

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

if( ret == SUCCESS )

printf("Files is successfully Remove\n");

else

DisplayError(ret);

continue;

}

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

{

ret = StatFile( command[1] );

if( ret != SUCCESS )

DisplayError(ret);

continue;

}

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

{

ret = FstatFile( atoi(command[1]) );

if( ret != SUCCESS )

DisplayError(ret);

continue;

}

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

{

fd = GetFDFromName(command[1]);

if( fd == ERR\_FILE\_NOT\_FOUND )

{

DisplayError(fd);

continue;

}

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

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

ret = strlen(arr);

if( ret == 0 )

{

continue;

}

ret = WriteFile( fd , arr , ret );

if( ret != SUCCESS )

DisplayError(ret);

else

printf("Successfully write\n");

continue;

}

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

{

ret = truncateFile( command[1] );

if( ret != SUCCESS )

DisplayError(ret);

continue;

}

else

{

printf("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);

}

else

DisplayError(ret);

continue;

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

{

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

if( ret >= 0 )

{

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

}

else

DisplayError(ret);

continue;

}

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

{

fd = GetFDFromName(command[1]);

if( fd == ERR\_FILE\_NOT\_FOUND )

{

DisplayError(fd);

continue;

}

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

if( ptr == NULL )

{

printf("Error\n");

continue;

}

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

if( ret > 0 )

\_write(2,ptr,ret);

else

DisplayError(ret);

continue;

}

else

{

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

continue;

}

}else if( count == 4)

{

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

{

fd = GetFDFromName(command[1]);

if( fd == ERR\_FILE\_NOT\_FOUND )

{

DisplayError(fd);

continue;

}

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

if( ret == SUCCESS )

{

printf("Successfullt seek\n");

}

else

DisplayError(ret);

continue;

}

else

{

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

continue;

}

}else

{

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

continue;

}

}

}