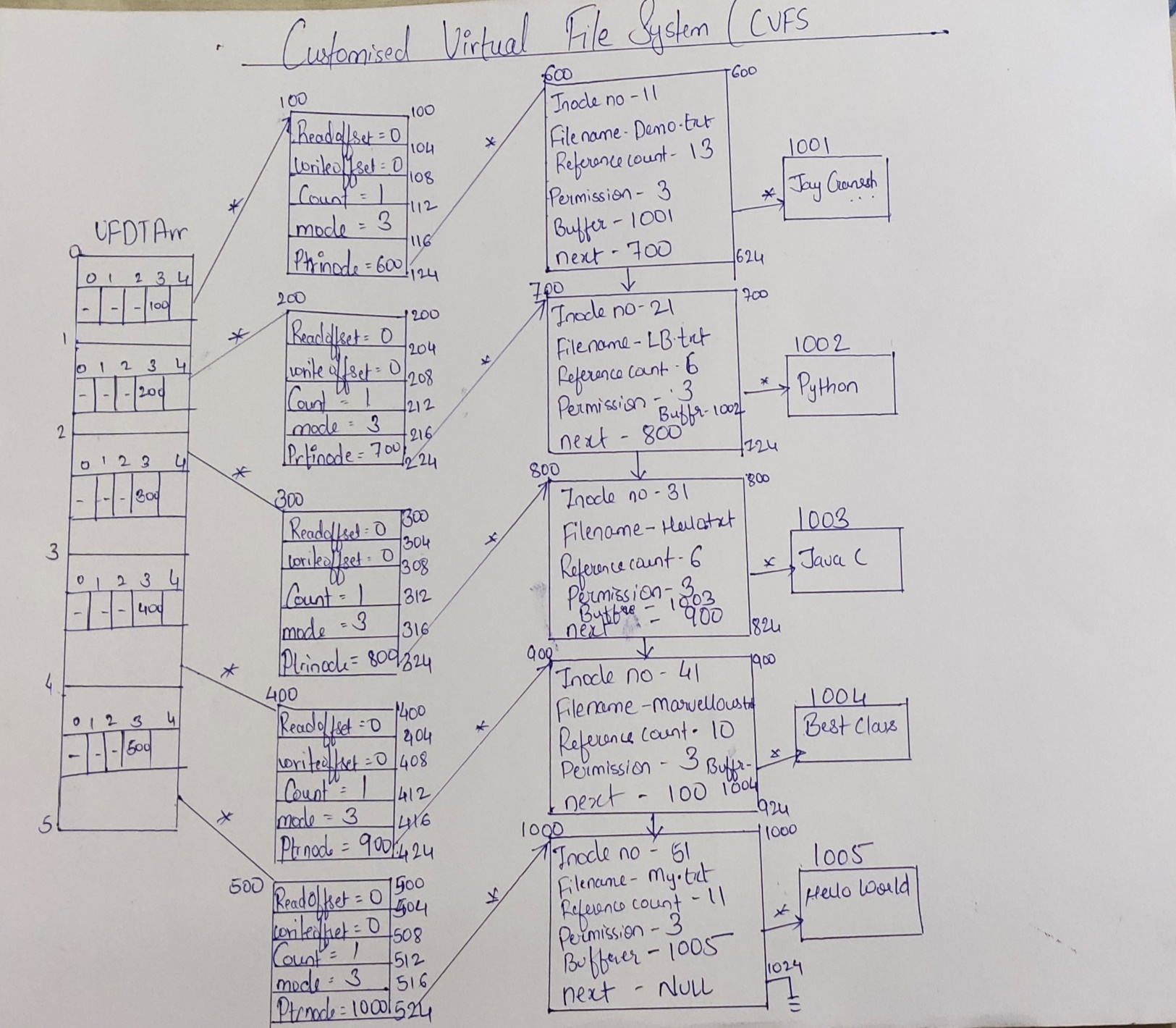
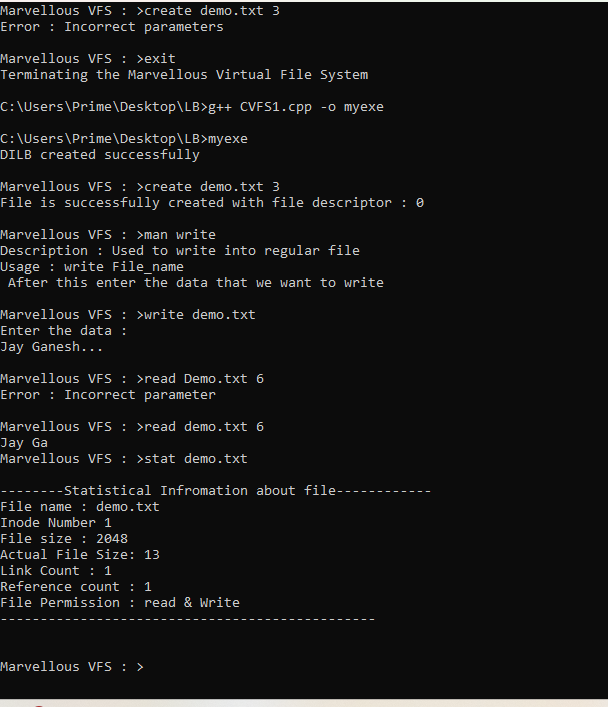
* **Customized Virtual File System**
* Technology used –
  + System Programming Using C User interface used
* Platform required -
  + Windows Platform Or Linux Distributions.
* Hardware requirements – N.A
* Description of the project –
* In This Project I Emulate All Data Structures Which Are Used By Operating System To Manage File System Oriented Tasks.
* As The Name Suggests Its Virtual Because I Maintain All The Records In Primary Storage (RAM).
* In This Project I Create All Data Structures Which Required For File Subsystems As : FT, UAREA, UFDT, SB, DILB, DB etc.
* I Provide All Implementations Of Necessary System Calls And Commands Of File subsystem As : Open, Close, Read, Write, Lseek, Create, RM, LS, Stat, Fstat etc.
* While Creating This Project All Above Functionality I Use Our Own Data Structures By Referring Algorithms of UNIX Operating System
* By Using This Project I Can Get Overview Of UFS (UNIX File System) On Any Platform.
* Data structures used in the project-
  + Singly Linear Linked List
* Diagram of data structures used in the project



* **The flow of the project -**
* 1.To create virtual file system I have used customized format as
* Firstly I have defined User defined macro’s in it to access function runtime as MAXNODE, READ, MAXFILESIZE, WRITE, REGULAR, SPECIAL, START, CURRENT, END.
* 2. Implemented Super block to Keep all the records of Total no. of blocks, free blocks, Total no. of inodes, free inodes, size of file system.
* 3. Than implemented Inode – Inode is a structure which contains complete information about file. Every file contains its unique inode.
* 4. Implemented DILB(Disk Inode List block) – It contains the Linked-List of all the inodes.
* 5. Implemented File Table – It is a table which contains information about the file that we want to open, file table contains entities like Offset, Modes, Pointers.
* 6.Implemented UAREA- It’s a specific area allocated by the operating system to the process, which contains the important information about the process. For every process there is separate UREA.
* 7. Implemented UFDT- It’s a array of pointers which holds the address of specific entry from file table. First 3 entries from UFDT are reserved for standard input, standard output, and standard error.
* 8.Implemented Incore Inode table- It’s a table which contains the Inodes which are loaded into the memory. All file’s inodes are present IIT. The complete information of Inode is stored inside IIT.
* Actual 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 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;   // 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 data from regular file\n");
* printf("Usage : read File\_name No\_Of\_Bytes\_To\_Read\n");
* }
* else if(strcmp(name,"write")==0)
* {
* printf("Description : Used to write into regular file\n");
* printf("Usage : write File\_name\n After this enter the data that we want to write\n");
* }
* else if(strcmp(name,"ls")==0)
* {
* printf("Description : Used to list all information of files\n");
* printf("Usage : ls\n");
* }
* else if(strcmp(name,"stat")==0)
* {
* printf("Description : Used to display information of file\n");
* printf("Usage : stat File\_name\n");
* }
* else if(strcmp(name,"fstat") == 0)
* {
* printf("Description : Used to display information of file\n");
* printf("Usage : stat File\_Descriptor\n");
* }
* else if(strcmp(name,"truncate")==0)
* {
* printf("Description : Used to remove 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 opened file\n");
* printf("Usage : close File\_name\n");
* }
* else if(strcmp(name,"closeall")==0)
* {
* printf("Description : Used to close all opened file\n");
* printf("Usage : closeall\n");
* }
* else if(strcmp(name,"lseek")==0)
* {
* printf("Description : Used to change file offset\n");
* printf("Usage : lseek File\_Name ChangeInOffset StartPoint\n");
* }
* else if(strcmp(name,"rm")==0)
* {
* printf("Description : Used to delete the file\n");
* printf("Usage : rm File\_Nmae\n");
* }
* else
* {
* printf("ERROR : No manual entry available \n");
* }
* }
* //It displays all System calls and its uses
* void DisplayHelp()
* {
* printf("ls : To List out all file\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");
* }
* //It holds addr of specific entries from file table
* 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;
* }
* //creates LL of inodes in DILB
* 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 successfully\n");
* }
* //keeps all info of total inodes and free inodes
* void InitialiseSuperBlock()
* {
* int i = 0;
* while(i < MAXINODE)
* {
* UFDTArr[i].ptrfiletable = NULL;
* i++;
* }
* SUPERBLOCKobj.TotalInodes = MAXINODE;
* SUPERBLOCKobj.FreeInode = MAXINODE;
* }
* //create file function
* 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->ptrnode->Buffer);
* free(UFDTArr[fd].ptrfiletable);
* }
* UFDTArr[fd].ptrfiletable = NULL;
* (SUPERBLOCKobj.FreeInode)++;
* }
* //reads data from file
* 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;
* }
* //write into file
* 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;
* }
* //opens file with modes
* 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)--;
* }
* //close file
* 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;
* }
* //close all existing files
* void CloseAllFile()
* {
* int i = 0;
* while(i < 50)
* {
* if(UFDTArr[i].ptrfiletable != NULL)
* {
* UFDTArr[i].ptrfiletable -> readoffset = 0;
* UFDTArr[i].ptrfiletable -> writeoffset = 0;
* (UFDTArr[i].ptrfiletable -> ptrinode -> ReferenceCount)--;
* break;
* }
* i++;
* }
* }
* int LseekFile(int fd, int size, int from)
* {
* if((fd < 0) || (from > 2))
* return -1;
* if((UFDTArr[fd].ptrfiletable->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;
* }
* }
* }
* //list all existing files
* 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\t Inode number\t File size \t Link 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");
* }
* //this displays all information about file
* 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 Infromation 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;
* }
* //it removes all data from file
* 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();
* //shell
* while(1)
* {
* fflush(stdin);
* strcpy(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(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 memeory to write\n");
* if(ret == -3)
* printf("ERROR : It is not 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 parameters\n");
* if(ret == -2)
* printf("Error : There is no inodes\n");
* if(ret == -3)
* printf("Error : File already exists\n");
* if(ret == -4)
* printf("Error : Memory allocation failure\n");
* continue;
* }
* else if(strcmp(command[0],"open") == 0)
* {
* ret = OpenFile(command[1],atoi(command[2]));
* if(ret >= 0)
* printf("File is successfully opened with file descriptor : %d\n",ret);
* if(ret == -1)
* printf("Error : Incorrect parameters\n");
* if(ret == -2)
* printf("Error : File not present \n");
* if(ret == -3)
* printf("Error : Permission denied \n");
* continue;
* }
* else if(strcmp(command[0],"read") == 0)
* {
* fd = GetFDFromName(command[1]);
* if(fd == -1)
* {
* printf("Error : Incorrect parameter\n");
* continue;
* }
* ptr = (char \*)malloc(sizeof(atoi(command[2]))+1);
* if(ptr == NULL)
* {
* printf("Error : Memory allocation failure\n");
* continue;
* }
* ret = ReadFile(fd,ptr,atoi(command[2]));
* if(ret == -1)
* printf("ERROR : File not existing \n");
* if(ret == -2)
* printf("Error : Permission denied \n");
* if(ret == -3)
* printf("Error : Reached at 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("\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("\n Error : Command not found !!!\n");
* continue;
* }
* }
* return 0;
* }
* **Screenshot of output which demonstrates every feature** 
  + 
* What is mean by file system?
* Ans- 1.File System is way of storing, representing, the file in a particular format.
* 2.NTFS, FAT32, FAT64, UFS, are types of file system.
* 3.When we create any file system the previously stored data gets over-written and the new file system gets formed.
* 4.Every file system is divided into four parts: 1.Boot block 2.Super block. 3.Disk Inode List Block.4.Data Block
* Which file systems are used by Linux and Windows operating systems?
* Ans. NTFS, FAT32, FAT64, UFS, **exFAT, and EXT2/3/4**.are types of file system.
* What are the parts of the file system?
* Ans. Every file system is divided into four parts: 1.Boot block 2.Super block. 3.Disk Inode List Block.4.Data Block
* Explain UAREA and its contents.
* Ans: It’s a specific area allocated by the operating system to the process, which contains the important information about the process. For every process there is separate UAREA. It contain UFDT.
* Explain the use of the File Table and its contents.
* Ans: It is a table which contains information about the file that we want to open, file table contains entities like Offset, Modes, Pointers.
* Explain the use of InCore inode Table and its use.
* Ans. It’s a table which contains the Inodes which are loaded into the memory. All file’s inodes are present IIT. The complete information of Inode is stored inside IIT.
* What does inode mean?
* Ans. Inode is a structure which contains complete information about file. Every file contains its unique inode.
* What are the contents of Superblock?
* Ans: Super block to Keep all the records of Total no. of blocks, free blocks, Total no. of inodes, free inodes, size of file system.
* What are the types of files?
* Ans: NTFS, FAT32, FAT64, UFS.
* What are the contents of the inode?
* Ans: FileName, Inode Number, File Size, File ActualSize, File Type, Buffer, LinkCount, ReferencceCount, Permission, next.
* What is the use of a directory file?
* Ans: Directory is container which store files and its inodes.
* How does the operating system maintain security for files?
* Ans: Operating systems provide several file protection features, including **file permissions, encryption, access control lists, auditing, and physical file security**.
* These measures allow administrators to manage access to files, determine who can access them, what actions can be performed on them, and how they are stored and backed up.
* What happens when a user wants to open the file?
* Ans: When you **open** a file, it follows the file extension and the directory. Each file contains an inode number and file name. An inode number is an identification number per file. Opening a file means activating a file descriptor that follows the inode number associated with the file name.
* What happens when a user calls lseek system call?
* Ans**:** [**When a user calls lseek() system call, it repositions the read/write file offset**](https://www.bing.com/ck/a?!&&p=502a3d776c25d368JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczMA&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)[**1**](https://www.bing.com/ck/a?!&&p=debb9b0e3b930358JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczMQ&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)[**2**](https://www.bing.com/ck/a?!&&p=e2b100f5c1b8de24JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczMg&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly93d3cuZ2Vla3Nmb3JnZWVrcy5vcmcvbHNlZWstaW4tYy10by1yZWFkLXRoZS1hbHRlcm5hdGUtbnRoLWJ5dGUtYW5kLXdyaXRlLWl0LWluLWFub3RoZXItZmlsZS8&ntb=1)[**3**](https://www.bing.com/ck/a?!&&p=3fb49fe8f4aa1cd4JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczMw&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly93d3cudHV0b3JpYWxzcG9pbnQuY29tL3VuaXhfc3lzdGVtX2NhbGxzL2xsc2Vlay5odG0&ntb=1)**.**[**This means that it changes the positions of the read/write pointer within the file**](https://www.bing.com/ck/a?!&&p=831c659153a2bb61JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczNA&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)[**1**](https://www.bing.com/ck/a?!&&p=b49e28dbb77f1c57JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczNQ&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)**.**[**Any read or write operations happen at the position pointed to by the pointer, and lseek() system call helps to manage the position of this pointer within a file**](https://www.bing.com/ck/a?!&&p=7016d360fb0b40c6JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczNg&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)[**1**](https://www.bing.com/ck/a?!&&p=fd5cc313286b92f8JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczNw&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9kZXh0dXRvci5jb20vbHNlZWstc3lzdGVtLWNhbGwv&ntb=1)**.**[**Upon successful completion, lseek() returns the resulting offset location as measured in bytes from the beginning of the file**](https://www.bing.com/ck/a?!&&p=954c0f9a577cde18JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczOA&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9zdGFja292ZXJmbG93LmNvbS9xdWVzdGlvbnMvNDM2NjQ3MjUvc3VjY2Vzcy1vZi1sc2Vlay1zeXN0ZW0tY2FsbC1pbi1j&ntb=1)[**4**](https://www.bing.com/ck/a?!&&p=0699a5b813f564c8JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTczOQ&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9zdGFja292ZXJmbG93LmNvbS9xdWVzdGlvbnMvNDM2NjQ3MjUvc3VjY2Vzcy1vZi1sc2Vlay1zeXN0ZW0tY2FsbC1pbi1j&ntb=1)**.**[**Otherwise, a value of -1 is returned and erron is set to indicate the error**](https://www.bing.com/ck/a?!&&p=5e0f630c182dc08cJmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTc0MA&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9zdGFja292ZXJmbG93LmNvbS9xdWVzdGlvbnMvNDM2NjQ3MjUvc3VjY2Vzcy1vZi1sc2Vlay1zeXN0ZW0tY2FsbC1pbi1j&ntb=1)[**4**](https://www.bing.com/ck/a?!&&p=287e59f259357310JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTc0MQ&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=What+happens+when+a+user+calls+lseek+system+call%3f&u=a1aHR0cHM6Ly9zdGFja292ZXJmbG93LmNvbS9xdWVzdGlvbnMvNDM2NjQ3MjUvc3VjY2Vzcy1vZi1sc2Vlay1zeXN0ZW0tY2FsbC1pbi1j&ntb=1).
* What is the difference between library function and system call?
* Ans: A system call is a function provided by the kernel to enter into the kernel mode to access the hardware resources.
* A Library call is a function provided by the programming library to perform a task.
* What is the use of this project?
* Ans: **Implementations Of Necessary System Calls And Commands Of File subsystem As: Open, Close, Read, Write, Lseek, Create, RM, LS, Stat, Fstat etc.**
  + **Creating This Project All Above Functionality I Use Our Own Data Structures By Referring Algorithms of UNIX Operating System.**
* Is there any improvement needed in this project?
* Ans: I will work on expanding its features and improvements.
* **Explain the internal working of below system calls**
* **(Write the solutions in the documentation)**
* Open- Open file for read or write into file.
* Close- Used to close the opened file.
* read – It will read the data from file and it read the given number of characters from file.
* write- After this command we will write contents into the file.
* lseek – **lseek** () system call is used**to change the position of the read/write pointer within a file.**
* stat – Display all the statistical information the file.
* Chmod – Allows you to change the permission of file.
* unlink – used **to deletes a name from the filesystem**
* Add a screenshot of each above command which demonstrates its use.
* **Explain use of below commands**
* **(Write the solutions in the documentation)**
* ls – lists all the files from directory
* ls – l is a Linux command that will list all “visible” files in current directory in long format and sorted in descending and alphabetical order
* ls – a **linux ls -a command, display all files and subdirectories in the current directory, including hidden**.
* rm – It deletes all files
* cat - If you have more than one file you want to view, you can specify multiple file names in your command. Their contents will be concatenated and sent to your screen
* cd- it changes directory
* chmod – it allows to change permission of file
* cp - The cp(copy) command is used to copy files or directories.
* df - **df** shows the disk space in **1 K** blocks
* find -
* grep -  use **grep** to search log files for specific entries, locate variables and functions within [codebases](https://phoenixnap.com/glossary/what-is-a-codebase), and detect system-related issues.
* ln -  command to create the links for the files
* mkdir - The mkdir(make directory) command is used to create a new directory.
* pwd - The pwd(print working directory) command is used to print out the current directory we are in.
* touch -  Create a new file or update its timestamp.
* uname- [The struct will contain information such as the kernel name and system node name](https://www.bing.com/ck/a?!&&p=64186b12fb11c635JmltdHM9MTcyMDU2OTYwMCZpZ3VpZD0wNGQ4NzE1Yi0yNjM4LTY1M2YtMzgxZC02Mjg5Mjc4YTY0N2ImaW5zaWQ9NTgyOQ&ptn=3&ver=2&hsh=3&fclid=04d8715b-2638-653f-381d-6289278a647b&psq=uname+system+call+&u=a1aHR0cHM6Ly93d3cuaW5mb3JtaXQuY29tL2FydGljbGVzL2FydGljbGUuYXNweD9wPTIzNjE4JnNlcU51bT0xNg&ntb=1)
* stat – Display all statistical information of file
* man – used to display manual page for the given command.
* mkfs - **build a file system on a device**