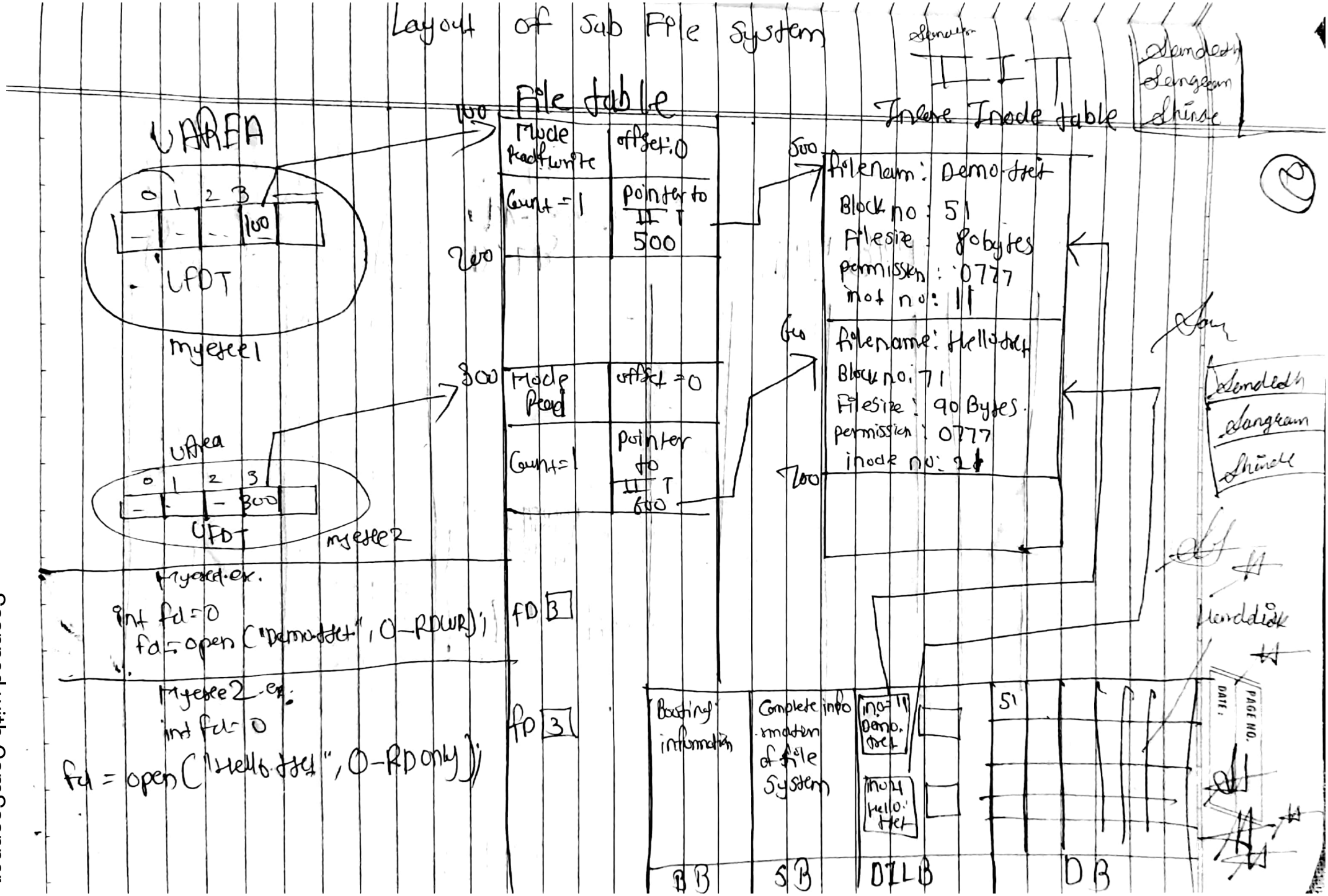
(1)

PAGE NO.
DATE :

Customised virtual File System (CVFS)

File System Layout



(3) All Structure in CVFS

PAGE NO.
DATE :

Superblock — SUPERBLOCK, *PSUPERBLOCK;

5	TotalInodes
5	FreeInodes

:

inode — INODE, *PINODE, **PPINODE

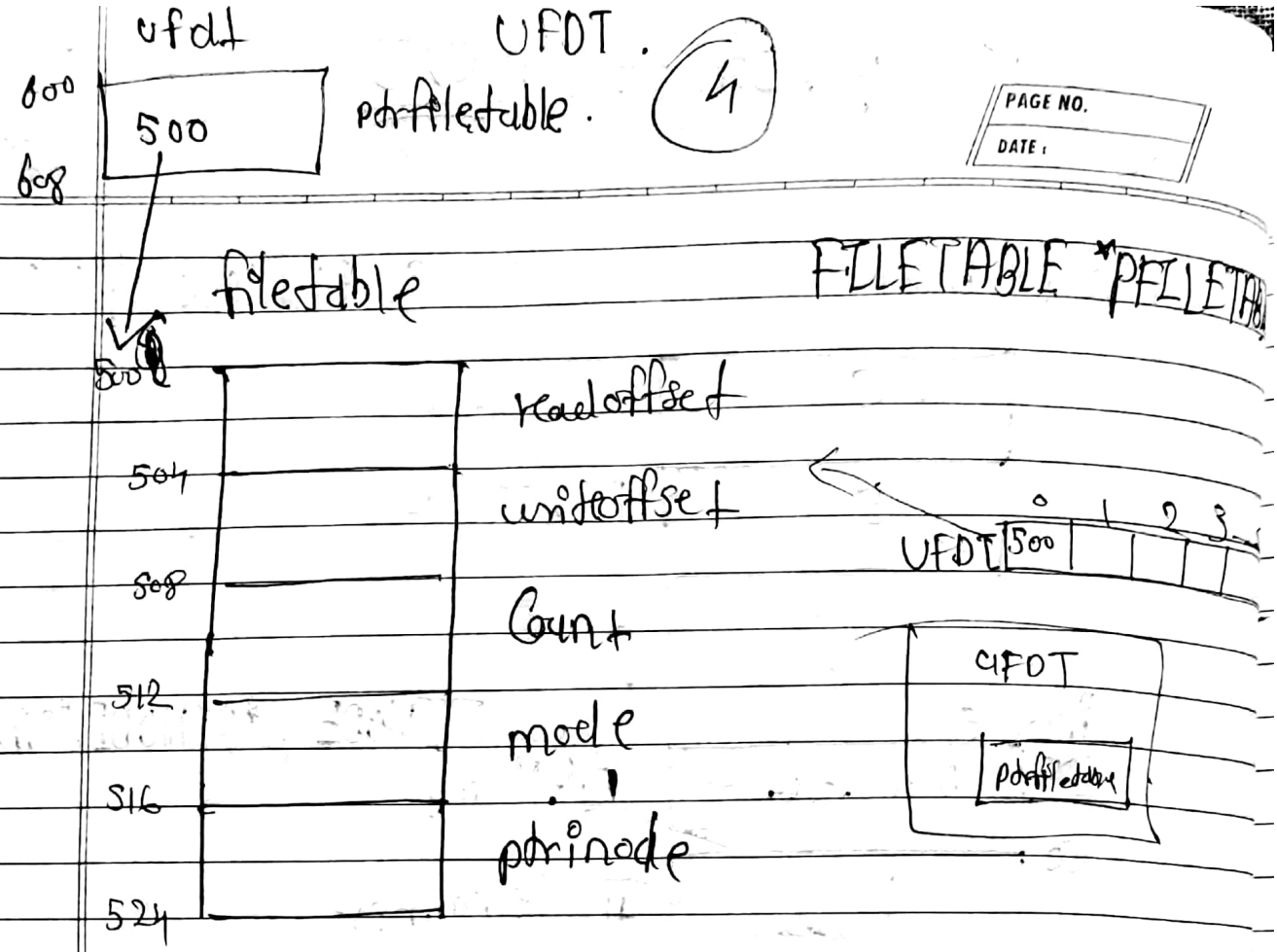
100 Demo.txt	FileName
150 1	InodeNumber
154 1024	FileSize
158 13	FileActualSize
162 1(Regular)	FileType
166 1000	Buffer
174 1	LinkCount
178 ~~Refer~~ 1	ReferenceCount
182 777	permission
186 200	next

Jay Ganesh.

Same inode

300

Same inode



Global variables.

```
UFDT UFDTArr[50];
SUPERBLOCK SUPERBLOCKobj;
PINODE head = NULL.
```

* Macro :-

```
#define MAXINODE 50
#define READ 1
        write 2
MAXFILESIZE 1024

RFGULAR 1
SPECIAL 2
```

```
START 0
CURRENT 1
END 2
```

List of functions

(5) Sandesh Sangram Shinde

Commands Sanjay

PAGE NO.
DATE :

- ls :- to list created files
- man : for manual page. (man create)
- help :- to Displaying help.
- create to create file
- create filename.extension permission (1, 2, 3)

Read, write, RD & WR both.

All the functions used in CVFS

1. InitializeSuperblock ();
2. CreateDILB ;
3. ls_file ()
4. closeallFile();
5. DisplayHelp();
6. Stat_file (command [1])
7. fstat_file (atoi command [1])
8. CloseFileByName (command [1]) (fd Name)
9. rm_file (command [1]);
10. man (command [1])
11. GetFD fromName (command [1])
12. writeFile (fd, arr, ret);
13. truncate_file (command [1]);
14. CreatFile (Command [1], atoi (Command [2]));
15. openFile (com. 1, atoi com 2)
16. ReadFile (fd, ptr, atoi comm 2);
17. LseekFile (fd, atoi com 2, atoi com 3)
18. Get_INODE
19. FIA++ Str cmp()
20. GetFD From Name (char *Name)
21. memset
22. sscanf
23. atoi()
24. fflush
25. strncpy
26. fgets.

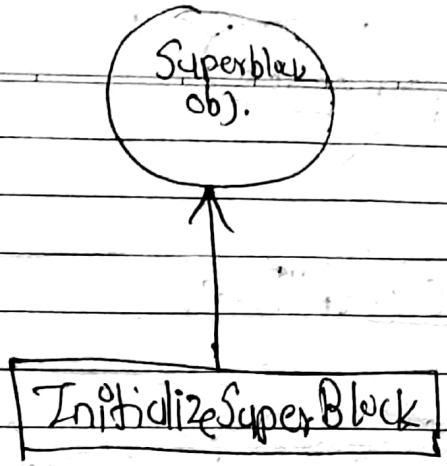
Scanned with CamScanner

6

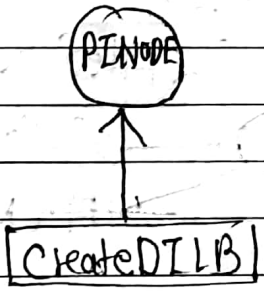
Diagrams

PAGE NO.
DATE :

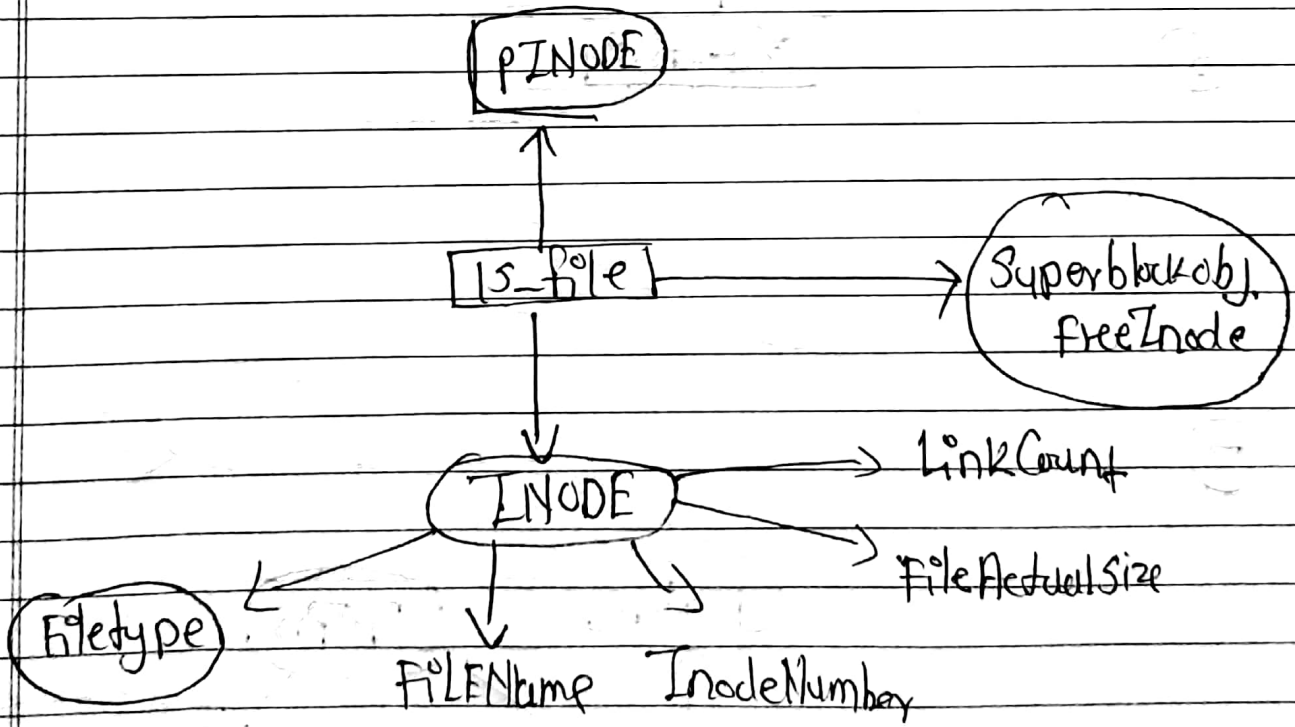
1



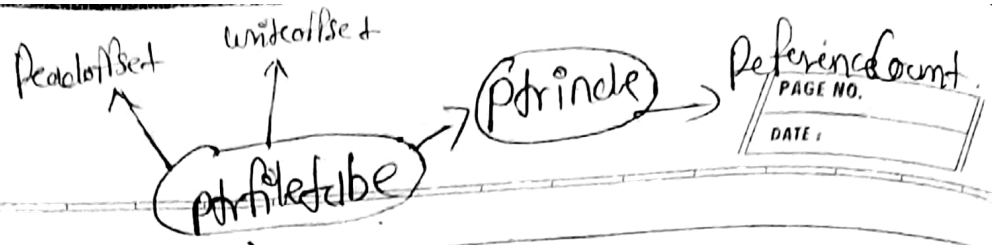
2



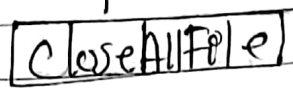
3



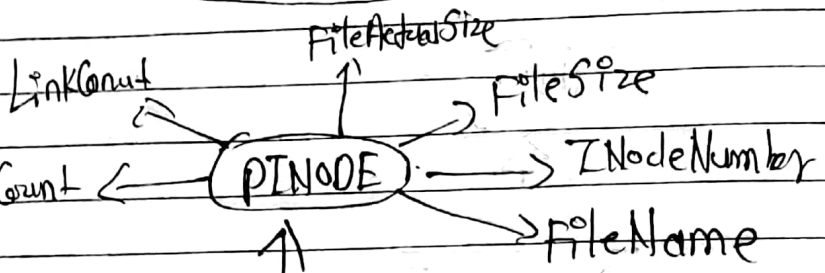
7



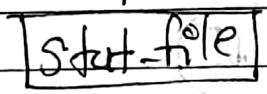
4



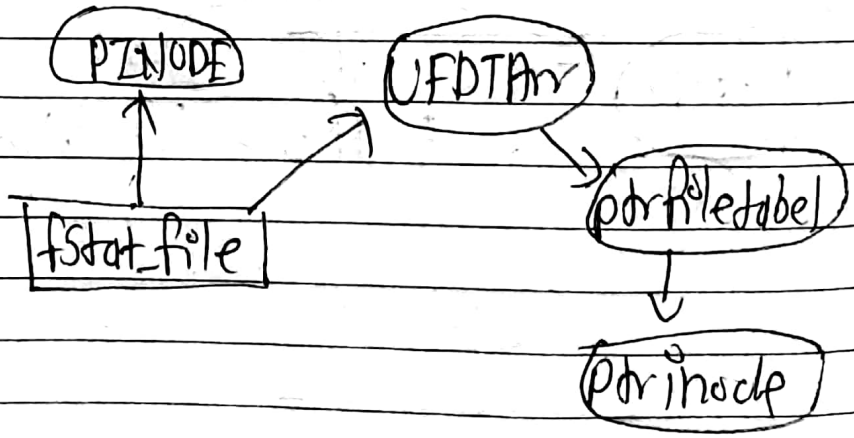
5 DisplayHelp — printf function.



6

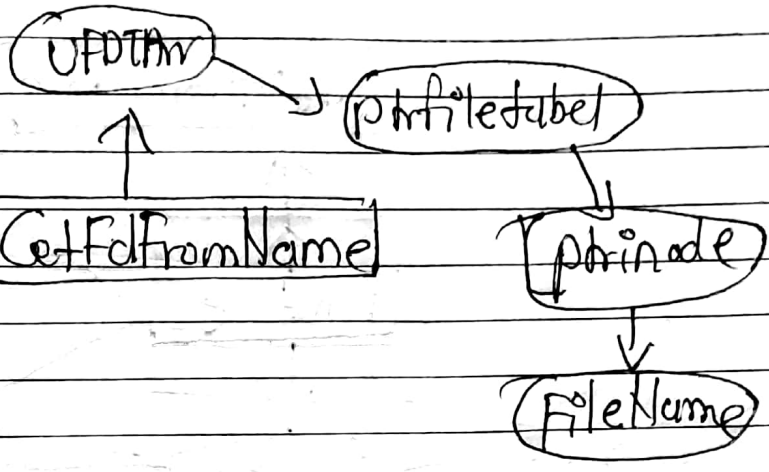


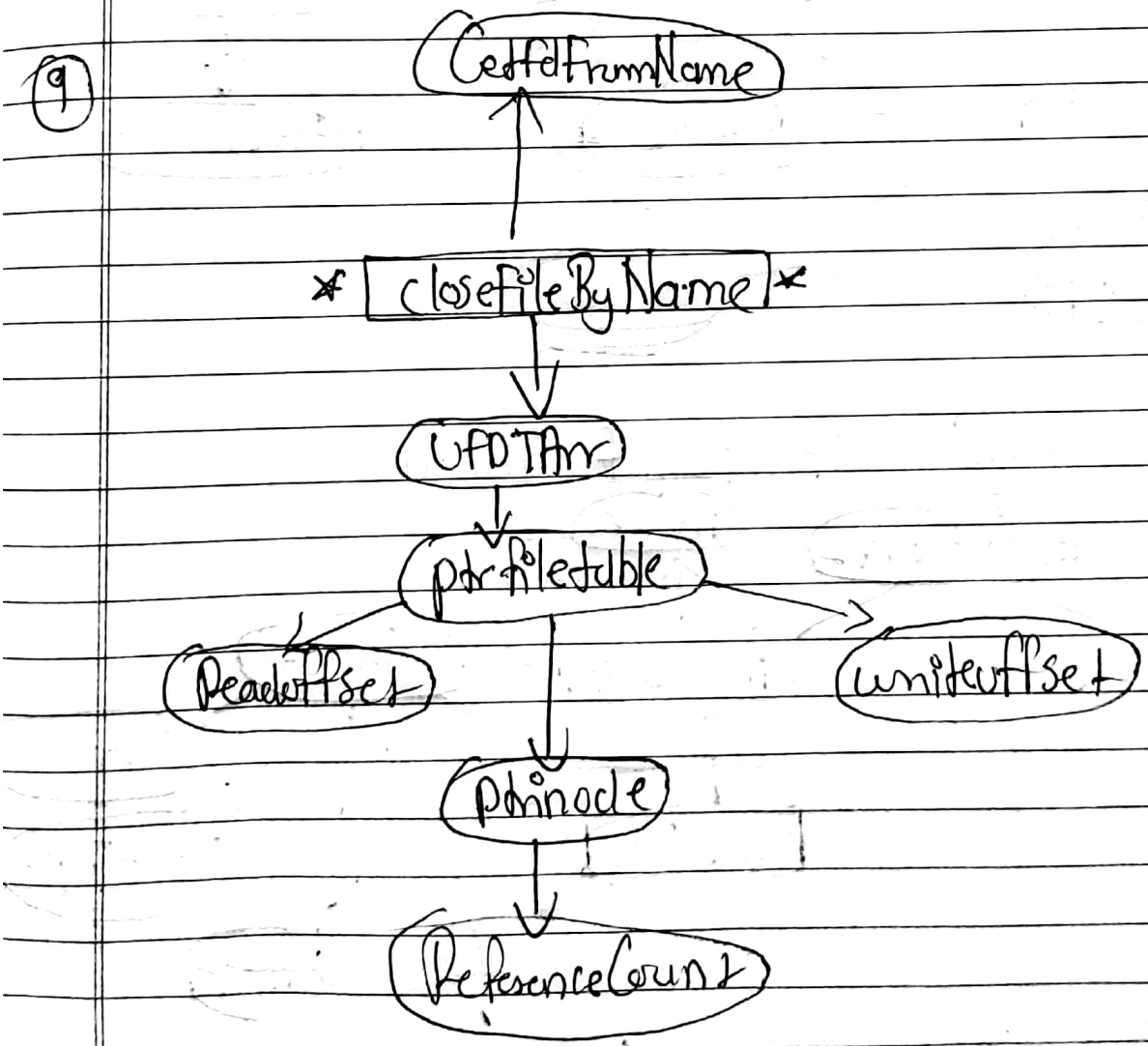
7



(8)

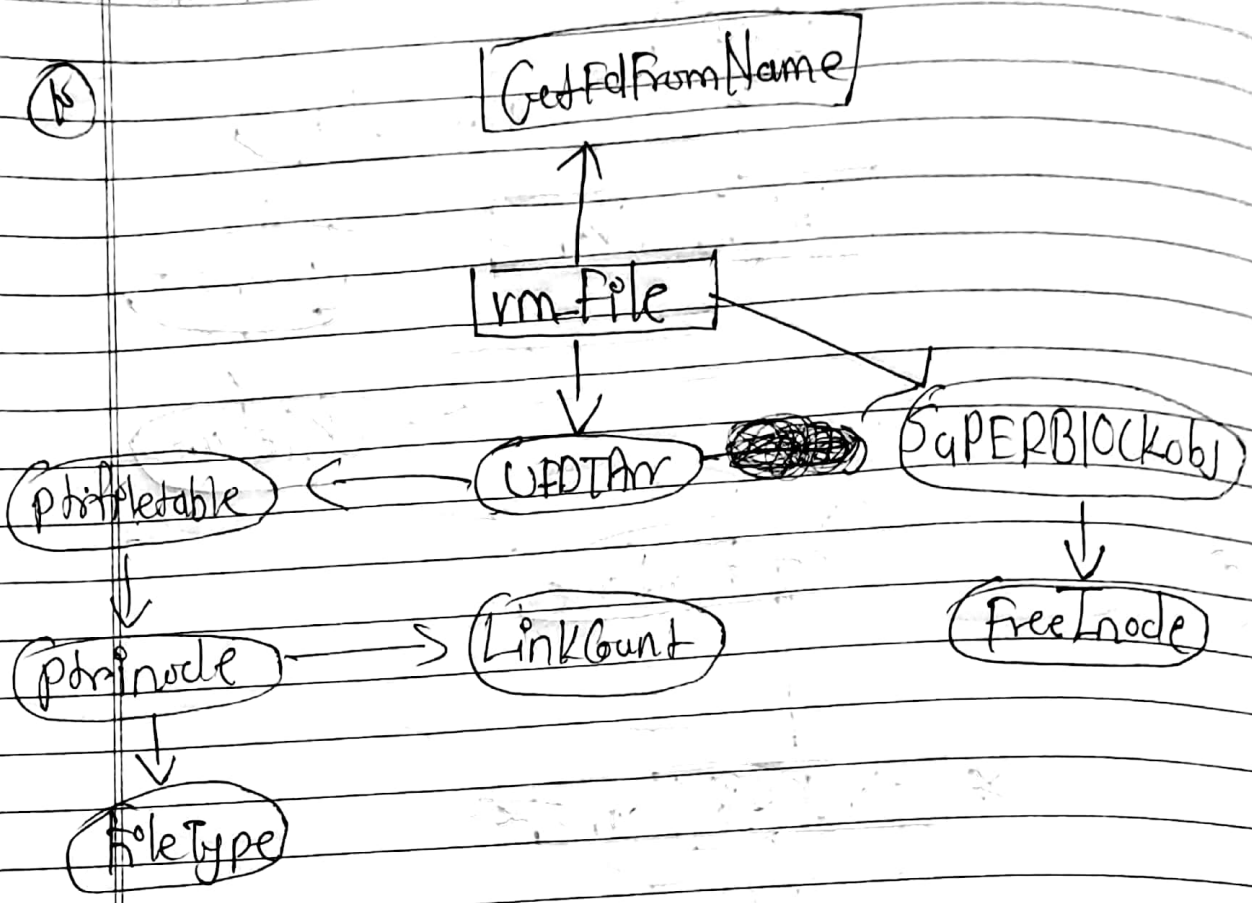
PAGE NO.
DATE :



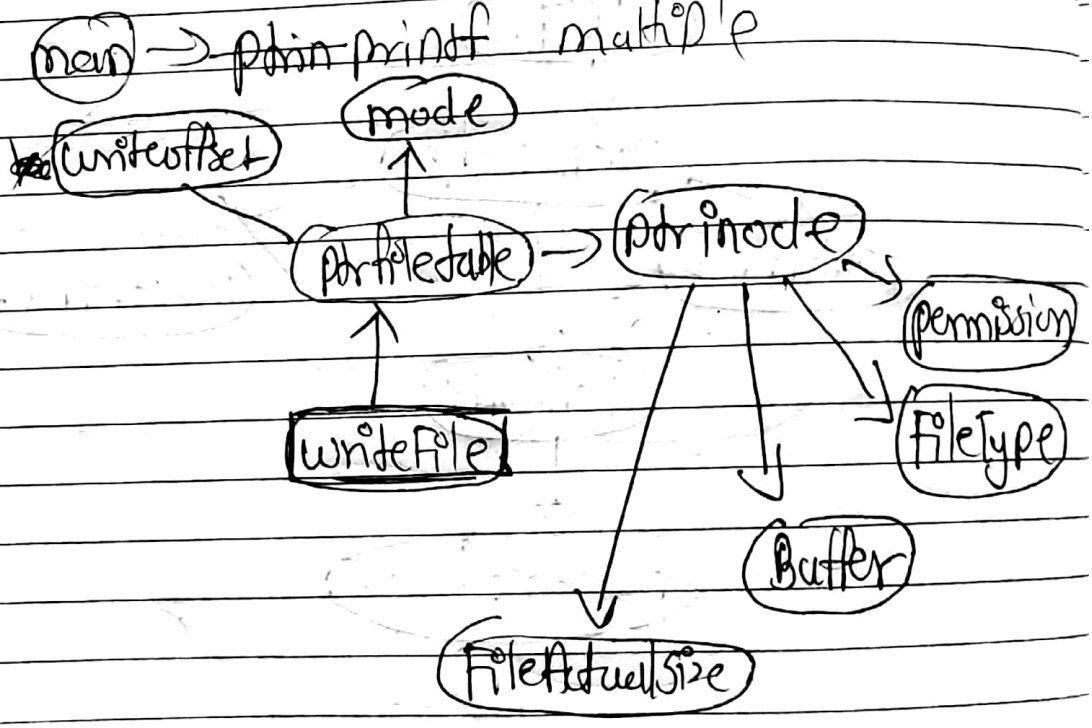


(9)

(10)

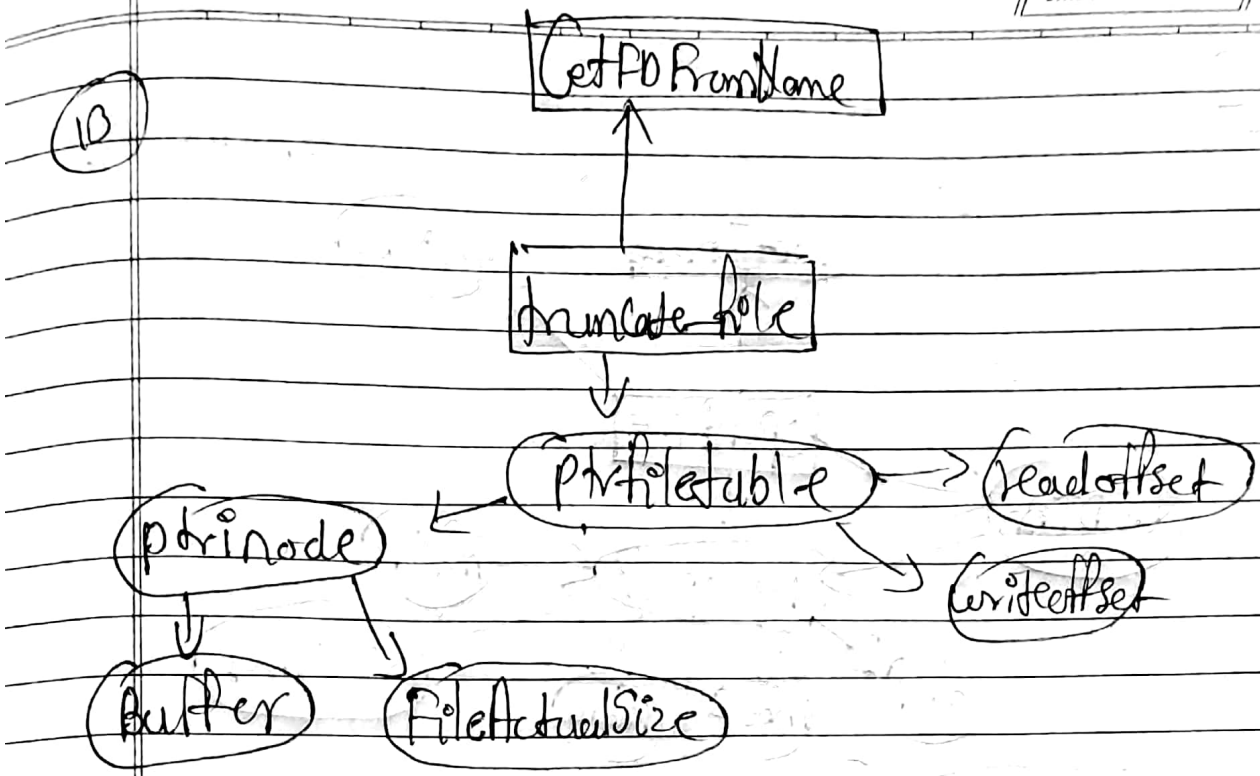


(11)

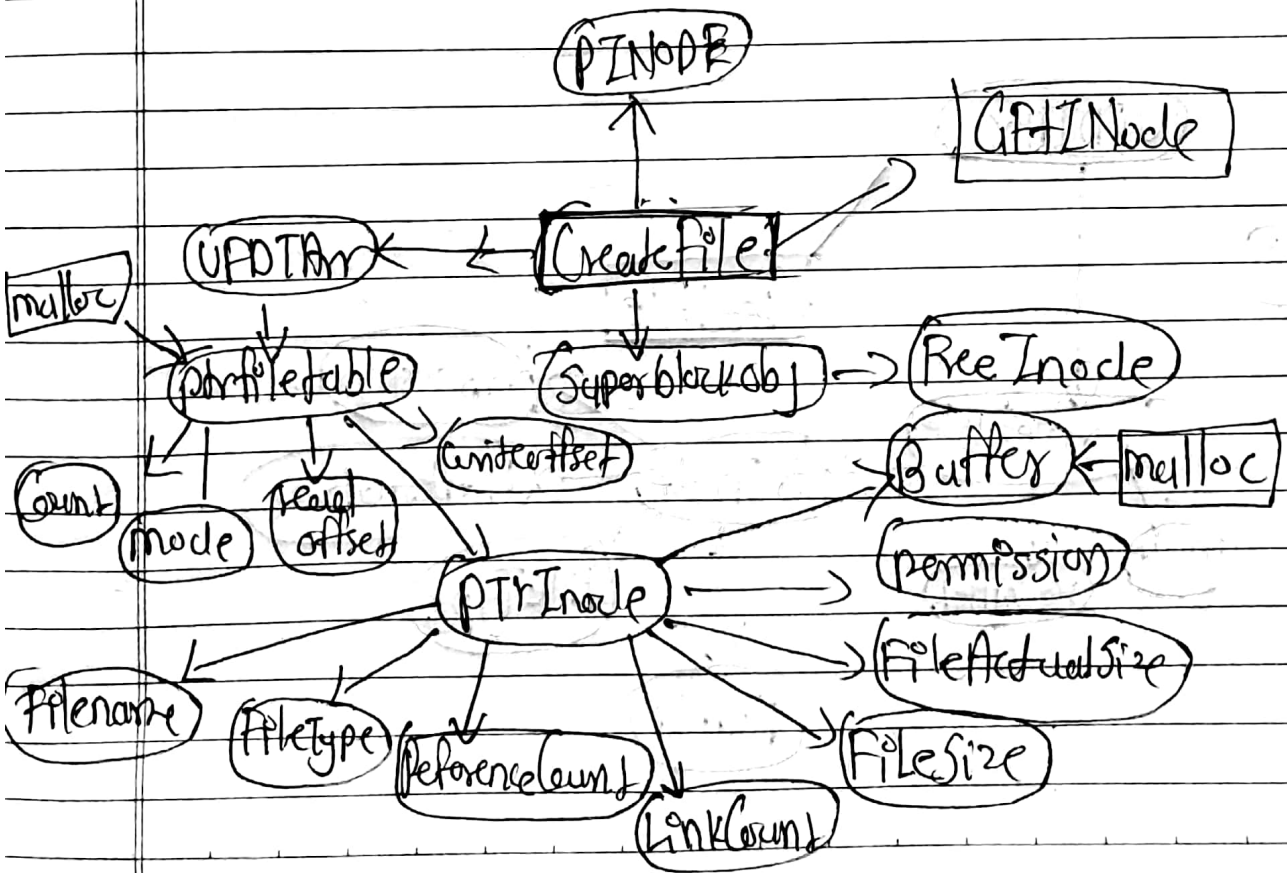


(12)

10

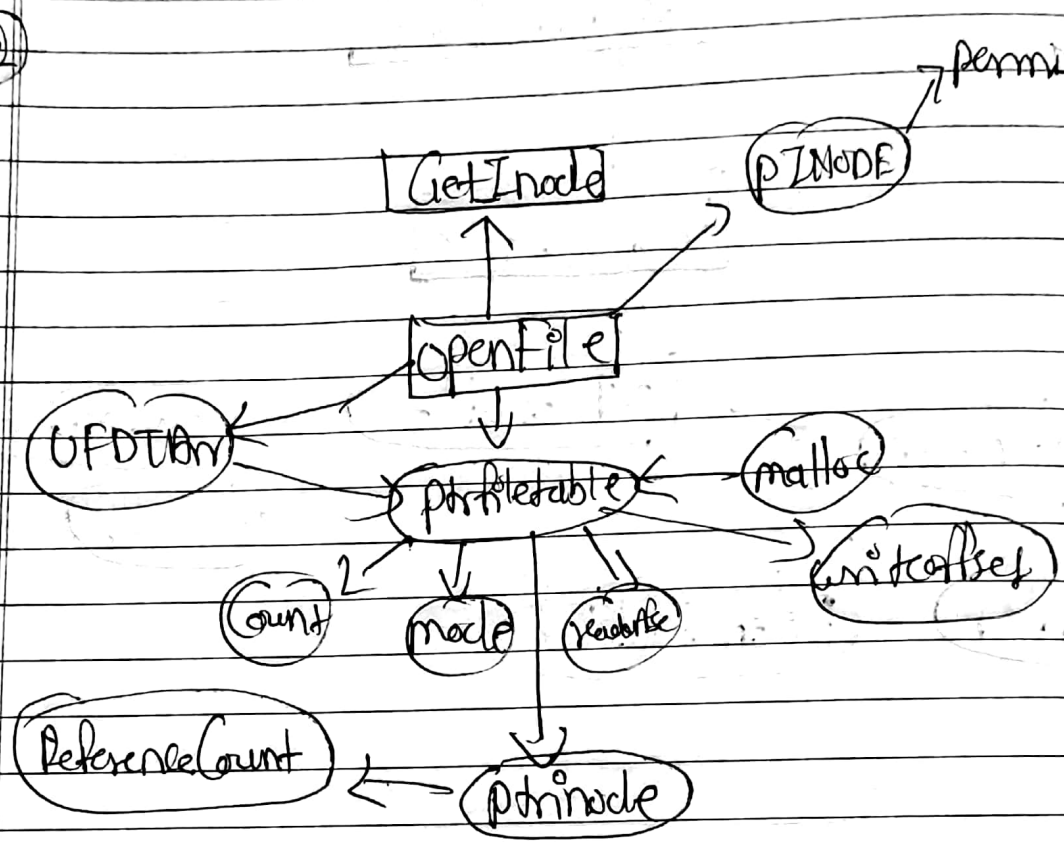


11

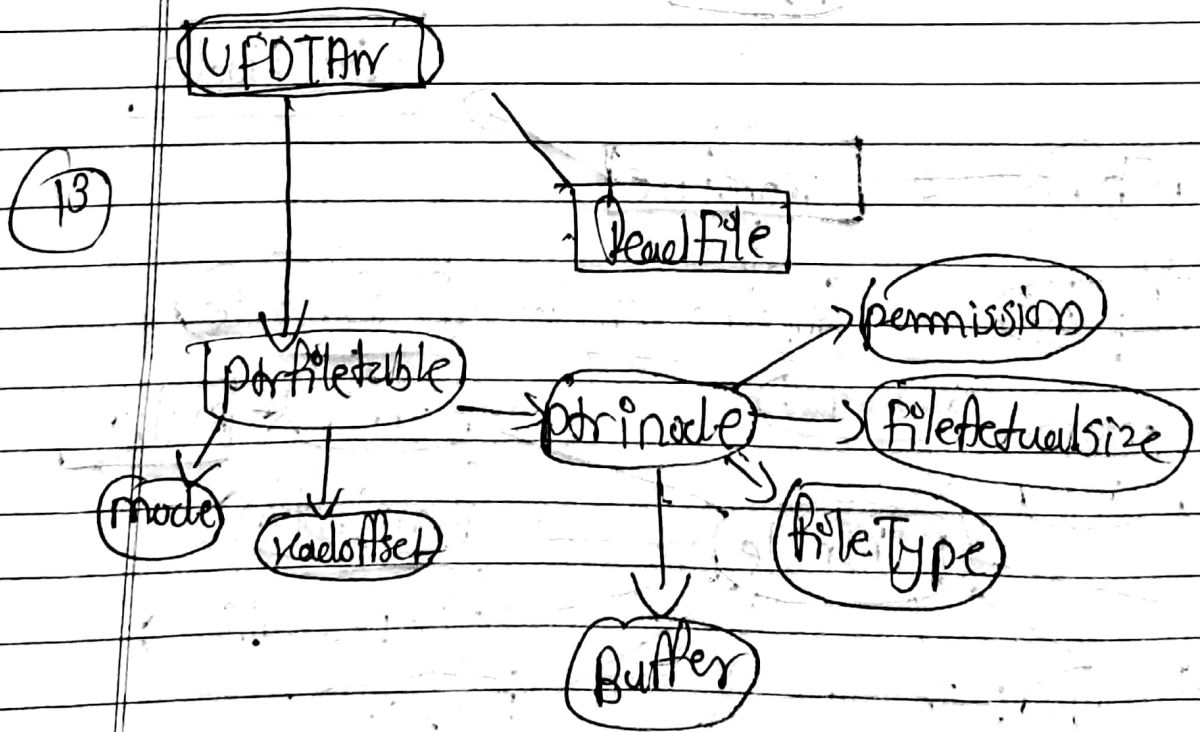


11

12

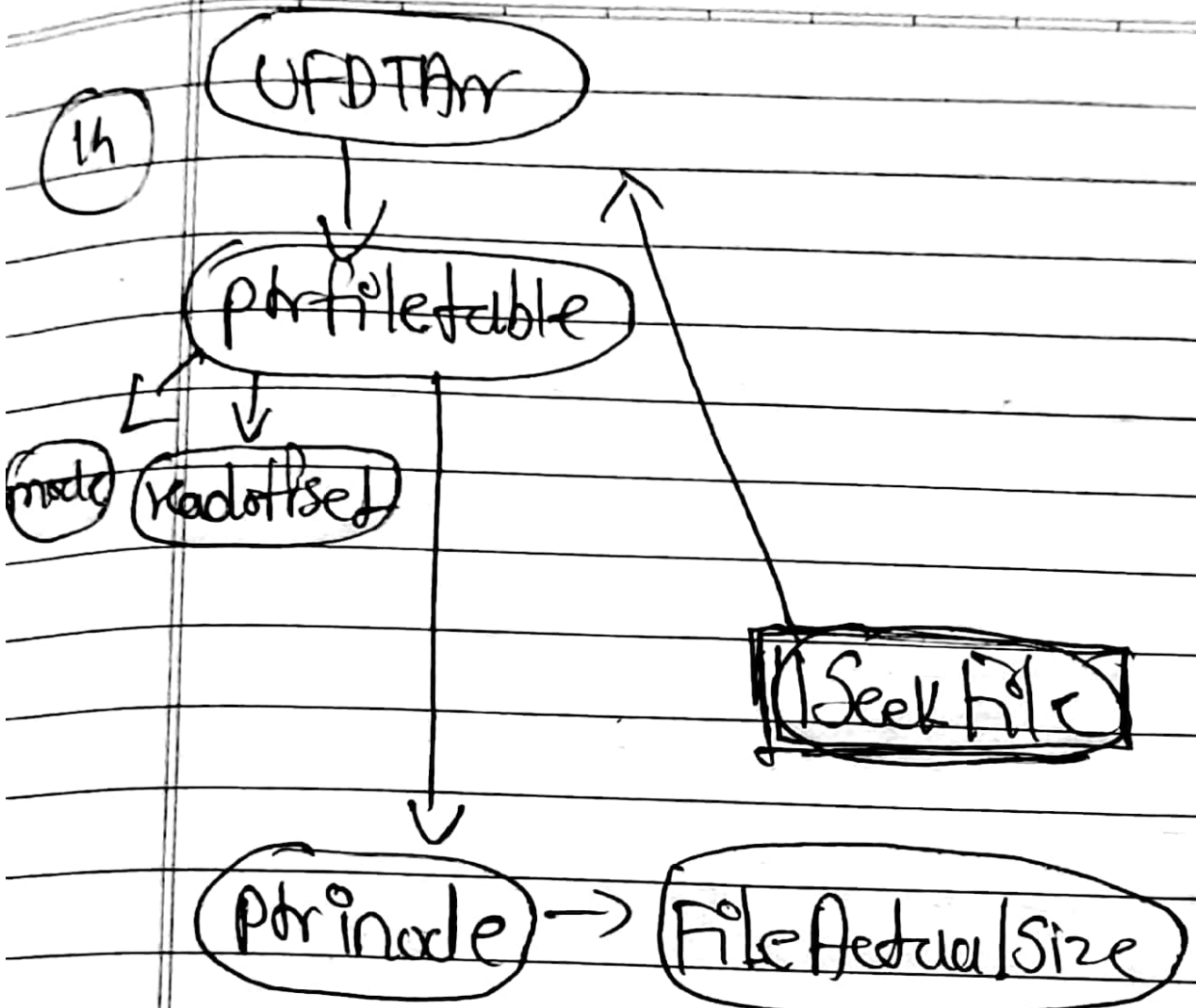


13



PAGE NO.

DATE :



(18)

functions

PAGE NO.
DATE :

1. InitialiSuperBlock();
2. CreateDILB();
3. ls_file();
4. closeAllFile()
5. DisplayHelp();
6. stat_file(char *name)
7. fstat_file(int fd)
8. CloseFileByName(int fd)
9. closeFileByName(char *name)
10. GetFdFromName(char *name)
11. rm_file(char *name)
12. man(char *name)
13. WriteFile(int fd, char *ar, int isize)
14. truncate_File(char *name)
15. CreateFile(char *name, int permission)
16. openfile(char *name int mode)
17. readfile(int fd, char ar*, int sz)
18. GetInode(char *name)
19. SeekFile(int fd, int size, int from)

(14)

Sandesh Gangaram Shinde
Sandesh Rama

PAGE NO.
DATE :

* Super block :- It is inventory of our filesystem.
It Contains 2 characteristics.
As (a) Total Inodes
(b) Free Inodes
while initializing the Superblock we are setting
Total Inodes = MaxInode (50);
Free Inodes = MaxInode (50);

* UFDT User File Descriptor table { This structure only Contains PFILE table pointer to file table

We have Created UFDT Named array with size 50;
while initializing Super block we have entered initialized the ef every block from 0 to 50 equal to NULL for pointer to file table.

(18) CreateFile(char *name, int permission)

PAGE NO.
DATE :

* CreateFile :-

① initialized i = 3 as it will be our filedescriptor.

② get the pInode temp = head:
in this line we get the head of inode which the first inde.

③ checked if any of the parameter is null or not valid : like name, permission=of permission >3
if it is
then return -1;

④ checked if there are any freeInodes by accessing Superblock of FMD.
in there are no any inodes left return -2
else

⑤ Decrement the freeInode of Superblock as we are going to take one.

⑥ Getting the Inode by passing the filename as a parameter to the GetInode function.
If there is any inode for our filename then return -3. which means file of that name already exists.

⑦ Searching for the Inode whose fileType == 0
if got then break:

⑧ Searching for UFDTArr[i].ptrfiletable = NULL a UFDTArray element whose pointer to ptrfiletable is NULL
If got then break
Same time increment the i to get the actual fd to return

(9) memory Allocations for the filetable for our newly created file and store the allocated memory locations address in our pointer to filetable'. Sor now.

(16) UFDT[i] will point to a filetable.

(c) In UFDTArr[i].ptrfiletable
  (a) Count = 1
  (b) mode = permission.
  (c) readoffset = 0;
  (d) writeoffset = 0;

(17) (e) UFDTArr[i].ptrfiletable -> ptrinode = temp
Some temp which have the address of that Inode whose Filesize is 0.
  (i) Change filetype to REGULAR (1)
  (ii) Reference Count = 1
  (iii) Link Count = 1
  (iv) Filesize = MAXFILESIZE (1024)
  (v) FileActualSize = 0; data is no written yet so 0.
  (vi) permissions = permissions.

(vii) Buffer = (char*) malloc (MAXFILESIZE)
Buffer is pointing to memory locations of size 1024 where our written file data will be store for this file.

(viii) returning fd means 1

* GetInode (char *name).

PAGE NO.
DATE :

(1) Getting pointer to inode temp = head;
if name = null return NULL;

(2) Travelling the LL of inodes till temp != NULL or we got the inode for our filename, returning that inode for our file.

* Void ls_file() *

(1) Initializing i=0 & then getting the head of the inode in temp i.e. temp = head;

(2) if Superblock.FreeInode == MaxINode return there are no file.
Because when FreeInode == MaxINode which means not a single file is created. that why we can't list them

(3) by using temp iterating whole LinkedList till the temp != NULL.
Inside the loop if Filetype is 0 we are not going print that file info. But if Filetype != 0 then we are going to print fileInfo that is FileName InodeNumber FileActualSize LinkCount.

PAGE NO.
DATE :

(17)

* (6) void CreateDILB()

(1) i=1 newn = NULL & temp = head.

(2) iterating the loop till i<=MAXINODE(50).

(3) Creating new 50 Inodes (new 50 Inodes) or Linked List of the 50 Inodes with all the default values.

(4) If temp == Null which mean linked list is empty ∴ Head = newn & temp = head

(5) else temp->next = newn ∴ temp now will point to the next new node.

(6) print (DILB) DISK Inode list block Created Succesfully.

It just Create all the empty Inodes linked list with default values.

(19)

PAGE NO.
DATE :

* void CloseAllFile()

(1) i = 0

(2) Iterating loop from 0 to 50 → MAXFILESIZE
i < 50

(3) if (UFDTArr[i].ptrfiletable != NULL)
The next procedure will happen only if the that element from UFDT has a filetable

(4) Setting the readoffset & writeoffset to 0 (Zero) for All the files which are created in our project.

(5) Decrementing the Reference Count for all the files present in our project.

(6) (UFDTArr[i].ptrfiletable->ptrinode->Reference(count)--

* void DisplayHelp();
prints of all the command and their usage for the enduser.

* void man (char *name);
man stands for Manual page.
This is Manual for every command its use and the usage of command.

* int Stat_file (char *Name)

① getting the Head of Inode LL in temp ∴ temp=head

② if name == NULL returns -1;

③ traverssing Complete Linked List of Inodes to find the node ~~which~~ of our file. by Comparing the Filename
if name is found then the loop will break. if name is not found which means loop will stop at temp=NULL;
∴ we will return -2 as file doesn't exists;

Sandesh Sangram Shinde

PAGE NO.
DATE :

Sandesh Sangram Shinde

(4) If file exists the we will have its Inode. by using the Inode we have all the Metadata of our file.
So will print all the Metadata of file.

(5) while printing the Metadata for permission we will print Read only for 1
Write only for 2
Read Write for 3.

Sandesh Sangram Shinde

PAGE NO.
DATE :

* int fstat_file(int fd)

The use of this function is same as the use of stat function just we are getting the information using fd.

① we have get the head of LL of inodes.

② if fd < 0 then the fd is wrong

③ UFDTArr[fd].ptrfiletable == NULL which means there is no file for given fd. or no any filetable, so we return -2 : which shows file Doesn't exists.

④ temp = UFDTArr[fd].ptrfiletable -> ptrinode;

In this step we are getting the inode of our file by travelling from UFDTArr to filetable to the Inode of our file

⑤ The next is same as stat_file function.

Vandeeth Sangeeam Shiv

PAGE NO.
DATE :

(23)

* CloseFileByName(char *name).

① initialize i=0, Getting FD using Name using GetFdFromName() function. fd will store in i:
if i==-1 then file doesn't exist.

② else.
Setting read & write offset to 0
UFDTArr[i].ptrfiletable.readoffset=0;
writeoffset=0;

③ Decrementing the Reference Count from the inode
(UFDTArr[i].ptrfiletable.ptrinode->ReferenceCount)--;

* closeFileByName(int fd).

① Setting read and write offset to 0 Zero.

② decreasing the reference Count.

PAGE NO.
DATE :

* int rm_File (char *name)

① Getting fd using name using GetFdFromNameFunc().

② if fd = -1 then file doesn't exists

③ Decrementing the LinkCount
(UFDTArr[fd].ptrfiletable->ptrinode->LinkCount)--;

④ Now if LinkCount is 0
changing filetype to 0
UFDTArr[fd].ptrfiletable->ptrinode->FileType=0;

⑤ in that deallocating the memory for filetable.
free(UFDTArr[fd].ptrfiletable);

⑥ initializing that fd on UFDT as NULL;
UFDTArr[fd].ptrfiletable = NULL;

⑦ modifying Superblock by notifying them that we have one more FreeInode,
(SuperblockObj.FreeInode)++;

Main use is we are removing that file

PAGE NO.
DATE :

int GetFdFromName(char *Name)

① we will iterate the loop 50 time (~~MAXFILESIZE~~ MAXINODES) to check if that filename is present in our linked list of Inodes i=0; i<50

② in loop if(UFDTArr[i].ptrfiletable != NULL
{ if strcmp(UFDTArr[i].ptrfiletable->ptrinode->Filename, Name)
== 0)
break;
}.

if the above if Condition we will Compare the filename and the name to be searched in we got it then break and will ~~return~~ the value of i thats the FD

else
if i == 50 the return -1 else return i

(26)

PAGE NO.
DATE :

* writeFile (int fd, char *arr, int size).

(1) 1st we will check
if UFDTArr[fd].ptrfiletable != WRITE
&& != READ+WRITE

then we will return -1; as the writing is not allowed in the file. file is not opened in write or Read+write mode.

(2) 2nd we will check

if UFDTArr[fd].ptrfiletable -> ptrinode -> permission
!= WRITE && != READ+WRITE

we will check if the file have a permission to write or Read+write else return -1 as writing not allowed

(3) 3rd we will check.

if the file's write offset is equal to the MAXFILESIZE bcz. will return -2; As there is no any sufficient memory.

(4) 4th we will check if file is not Regular then we return -3 as file is not Regular.

(5) After all this we will ~~leave~~ write in a fp

Scanned with CamScanner

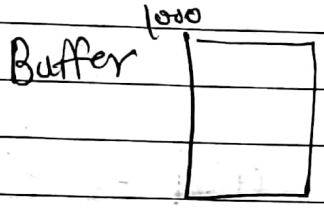
27

PAGE NO.
DATE :

6

① strncpy((ufOTA[fd].pfiletable->pinode->Buffer) + (ufOTA[fd].pfiletable->writeoffset), arr, isize);

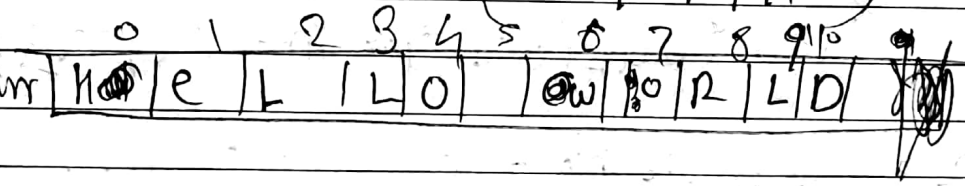
if our file has a nothing stored in it so,

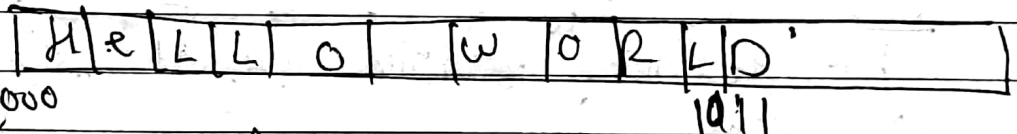


(Buffer + writeoffset)

e.g. (1000 + 0)

(1000, arr, 11)





writeoffset will be 11

② Now we will change the writeoffset will be the previous writeoffset + size

∴ writeoffset = writeoffset + 11

Ans: 11

③ then we will increase the fileactualsize. by →

Fileactualsize = File actualsize + isize

PAGE NO.
DATE :

④ then we will return "size" which represent how many member are written successfully.

memory Reset

* memset

memset function in c++ is used to fill a block of memory with a particular value.

Syntax

```
void *memset (void *ptr, int value, size_t num);
```

ptr :- pointer to the block of memory to fill.
value :- The value to be set, it is passed as an int but the memory is filled with its unsigned char representation.
num :- number of bytes to fill

return a pointer (i.e. the starting address of the memory block)

① initialize array to zero

```
int arr[10];
memset(arr, 0, sizeof(arr));
```

* truncate_file (char *name);

(29)

PAGE NO.

DATE :

① we will get the fd for our file name using GetFdFromName(name) function.
if fd == -1;
return -1; means file is not present.

② memset (UFDTArr[fd].ptrfiletable->ptrInode->Buffer, 0, 1024)

Starting address of Block to set to fill value | value to fill | no. of bytes to fill

using memset we are setting all 1024 bytes to 0. Think of this as deleting all the contents of the file from memory.
When all the bytes are set to zero therefore our file contains nothing.

③ Setting the read & write offset to 0.
UFDTArr[fd].ptrfiletable->ptrinode->read offset = 0
writeoffset = 0

④ modifying the actualfilesize to 0 zero

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

This function is used to clear all the data from the file.

Shiv   Sandesh Sangram

* OpenFile(char *name, int mode)

PAGE NO.
DATE :

(3)

① int i = 0; PINODE temp = NULL;

First. will check if Name == NULL or mode <= 0
return -1;
Incorrect Command

② temp = GetINode(Name);

if temp = NULL then File does not exist.

③ we will check if that file have a permission to open in Entered mode (i.e. Read, Write, R+W)
if not then return -3
means Not Allowed.

④ Now everything is clear.

⑤ we are going to find an empty slot in UFDTArray for our file. for that we will travel the Complete Array.

```
while (i<50)
{ if (UFDTArr[i].ptrfiletable == NULL)
    break;
  i++;
}
```

when we found an empty slot it will break and we will have an index of that empty slot as i.

PAGE NO.
DATE :

(6) allocating the memory for filetable.

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

(7) if unable to allocate memory return -1

if (UFDTArr[i].ptrfiletable == NULL)
return -1;

(8) we will initialize Count = 1; & mode = Enter mode

(9) then if the mode is Read & write
we will set Both read & writeoffset to 0;

UFDTArr[i].ptrfiletable -> writeoffset = 0
Readoffset = 0

(10) else if mode is only READ then will we just set readoffset to 0

UFDTArr[i].ptrfiletab -> Readoffset = 0;

(11) Same if mode is only write then will set only writeoffset to 0 Zero.

(12) At the end we put ptrinode the last member of filetable structure will point of the inode of the file which we have got at step 2

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

(13) increasing the Reference ~~pointer~~ ReferenceCount++;
Count value

PAGE NO.
DATE :

* Strncpy(dest, src, n) Copies n characters from src to dest
* if the src is shorter than n, the rest of dest is filled with \0,
* It does not null-terminate if src is longer than

* ReadFile (int fd, char *arr, int psize)

(33) EH = Error Handling.

(1) ~~EH~~ we set readSize = 0

(1) EH if there is no pofiletable return -1

(2) EH. if the mode of file in which it is opened in Not equal to Read or not equal to Readwrite so we Cant Read therefore return -2.

(3) EH. if the file permicdes not have permission to Read & write then return -2

UFDTArr[fd].pofiletable->ptrinode-> permission != READ + WRITE

return -2

(4) EH if readoffset is equal to the FilesActual size then return -3 because we have reached at the end of the file.

if(UFDTArr[fd].pofiletable -> readoffset == UFDTArr[fd].pofiletable -> ptrinode -> FileActualsize)

return -3

(5) EH If Filetype is not Regular then return -4

UFDTArr[fd].pofiletable->ptrinode ->FileType != Regular.;

return -4;

PAGE NO.
DATE :

(6) read size =
(UFDTArr[fd].ptrfiletable->ptrinode->fileActualSize - ..... - readoffset);

here we are calculating that how much can we read or no. of bytes available to read from current position (read offset) in the file until the end of file.

(7) if our read_size is less than required reading size then
we will copy the read_size data from the file into the Array to display and increase the readoffset in terms of read_size.

if (read_size < isize)
{
strncpy(arr, (UFDTArr[fd].ptrfiletably->ptrinode->Buffer) + (... readoffset), readsize);
}

(8) else as above same we will copy the isize no. of bytes from file and store it into the destination array.
and we will increase the readoffset in term of isize.

LSeekFile(int fd, int size, int from)

(35)

PAGE NO.
DATE :

① EH if fd<0 or from>2
return -1; means wrong value.

② EH if UFDTArr[fd].ptrfiletable == NULL; there means there no file present.
return -1

③ we check if the file is opend in READ mode or READ + WRITE
then
(a) if from == CURRENT

if((UFDTArr[fd].ptrfiletable -> readoffset) + size) > UFDTArr[fd].ptrfiletable -> ptrinode -> FileActualSize)
return -1.

This Condition will check that if readoffset + size is greater than file actualsize then return -1; as unable to perform that lseek.

if not exceed the aFileActualsize then readoffset will increase to size.

readoffset = readoffset + size;

(b) else if from == START

first will check if size is greater than FileActualSize will return -1 as unable to perform lseek
next if size < 0 then same as above.
At the end else
the readoffset will change to size.
readoffset = size;

PAGE NO.
DATE :

(c) if from == END

First we will check
if FileActualSize + size > MAXFILESIZE
yes then return -1
next
if readoffset + size < 0 then return -1
because some time size will be negative

end.
readoffset = FileActualSize + size;

(4) if File is opened in WRITE Mode

(a) if from == CURRENT.

EH if (writeoffset + size) > MAXFILESIZE
return -1.

EH if (writeoffset + size) < 0
return -1
unable to permit the lseek because index is going out of Bounds.

if (writeoffset + size) > (FileActualSize)
then
• FileActualSize = writeoffset + size;
else.
writeoffset = writeoffset + SI...

PAGE NO.
DATE :

if writeoffset +size is greater the filesActualsize.
then we will increase the file Actual size.
or = writeoffset +size.
else.
writeoffset will increased by size.

⑥ from == CurrentSTART.

if (size > MAXFileSize) or size < 0
return -1

if size is greater that our MAXFILESIZE or size < 0 the we will return -1.

eg. if size > FileActualSize)
then fileActualSize = size;
or
writeoffset = size;

⑦ from == END;

*EH if FileActualSize +size > MAXFILESize.
return -1.
because if the it short from End and plus size it will exceed the MAXFileSize.

*EH if writeoffset +size < 0 then return -1
because Index will be out of bound means we Cant perform the lseek operation.
.at the end,
writeoffset += FileActualSize +size;

SSCanf()

(33)

PAGE NO.
DATE :

* Used for parse space seperated strings (commands, inputs)

(2) Extract integers, floats, chars from a string.

(3) Split the string into tokens when not using strtok.

Eg. char str[] = "123 45.67 A";

int no; float f; char c;

SSCanf(str "%d %f %c", &a, &b, &c);

str — Source

"%d %f %c" — Pattern (format)

&a, &b, &c — assigning the space seperated value.

and it returns the number succesfully matched and assigned inputs

In our program,
we accept input in str

Count = SSCanf (str, "%s %s %s %s", Comnd, 2, 3, 4);

Count will be 4 if the inputs
Create file/del read write

✱ strcmp() ✱

(39)

PAGE NO.
DATE :

→ String Comparison

strcmp(str1, str2) Compare two strings.

If both are same return 0.

0 string equal
<0 if str1 < str2 lexicographically.
>0 if str1 > str2.

✱ atoi(str) ✱

atoi Converts a String to an integer.

ex. char str[] = "123";
int num = atoi(str) // num = 123

✱ fflush(stdin) ✱

fflush(stream) clears (flushes) the output buffer of a stream.

fflush(stdout) // forces output to appear immediately.

fflush(stdin) is used to clear input buffer

✱ fgets() ✱

∴ It reads a line of text from a file or stdin safely.

Syntax.

fgets(str, size, stream)

PAGE NO.
DATE :

fgets (str, size, stream);

str = destination array.
size = max characters to read (including \0)
stream : stdin for keyboard.

(4) LseekFile(int fd, int isize, int from)

EH :- Error Handling

(1) EH if fd < 0 of from is greater than 2 return -1 As invalid parameters.

(2) EH if there is no any filetable for that fd. then return -2; as file doesn't exists

(3) if file is open in Read or ReadWrite mode then

(A) if we want to perform the Lseek operation from Current offset then.

EH (i) if readoffset + size is exceeding the file Actual size then return -3

(ii) else we will ~~read~~ the increase the readoffset. by isize.

(B) if we want to perform the Lseek operation from the Start.

EH (i) we will check the is size is not if isize is greater than fileActual size will return -3 exceeding file size.

isize = change in offset

PAGE NO.
DATE :

(42)

EH (ii) if isize < 0 then return -1;

(iii) else we will change readoff set to isize,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
S	A	N	D	E	S	H		S	A	N	G	R	A	M		S	H	I	N	D	E

(after seek) ↑ ↑ Current offset.

from start

isize = 5.

readoffset = isize

(C) if we want to perform SEEK from end

EH (i) we will check if the summation of filesactual size and size. is greater than MAXFILESIZE (1024) if it is then return -3; exceeding file limit.

EH (ii) if Sum of readoffset + isize is less than zero then return -1.

(iii) else

readoffset will changed to filesactualsize + isize.

here isize could be negative.

because it is from = END.

PAGE NO.
DATE :

(4) if file is opened in write mode or Read + write.

(A) if from == Current

EH (i) writeoffset is greater than MAXFILESIZE return -3;

EH (ii) if writeoffset + isize is < 0 return -3

(iii) if writeoffset + isize > Filefactualsize

then Filefactualsize = writeoffset + isize.

(iv) else writeoffset = writeoffset + isize.

(B) from == Start

EH (i) isize is greater than MAXFILESIZE or size < 0. then return -3.
File sizelimit exceeding.

(ii) else if
isize > Filefactualsize.
then
filefactualsize = isize.

(iii) else
writeoffset = isize.

PAGE NO.
DATE :

(c) from == END.

EH (i) if fileActual > maxfilesize.
return -3.

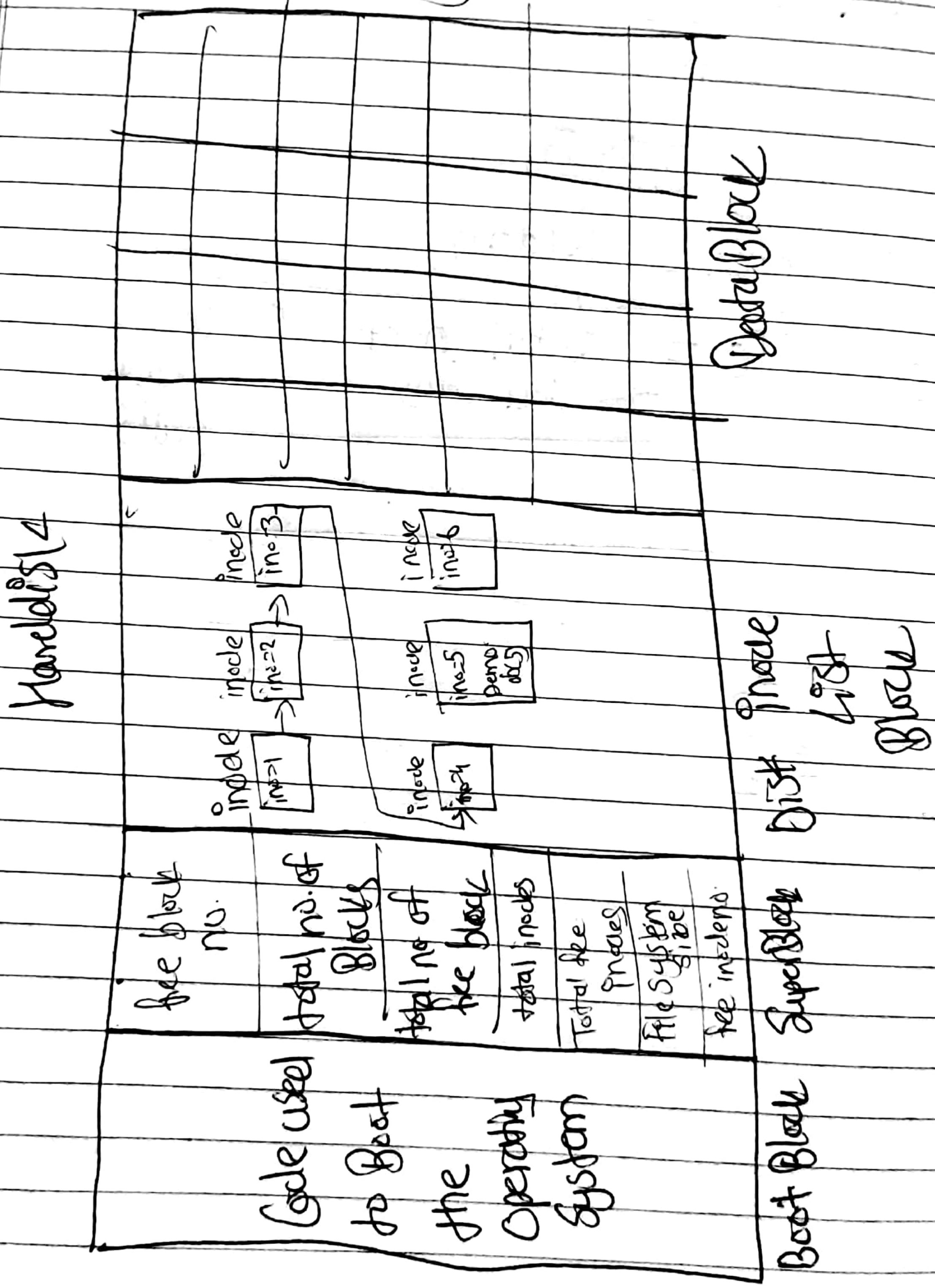
EH (ii) if writeoffset + size < 0
return -3.

(iii) writeoffset = fileActualsize + size;

①

Customised virtual file system (CVFS)

File System Layout



Layout of Sub File System