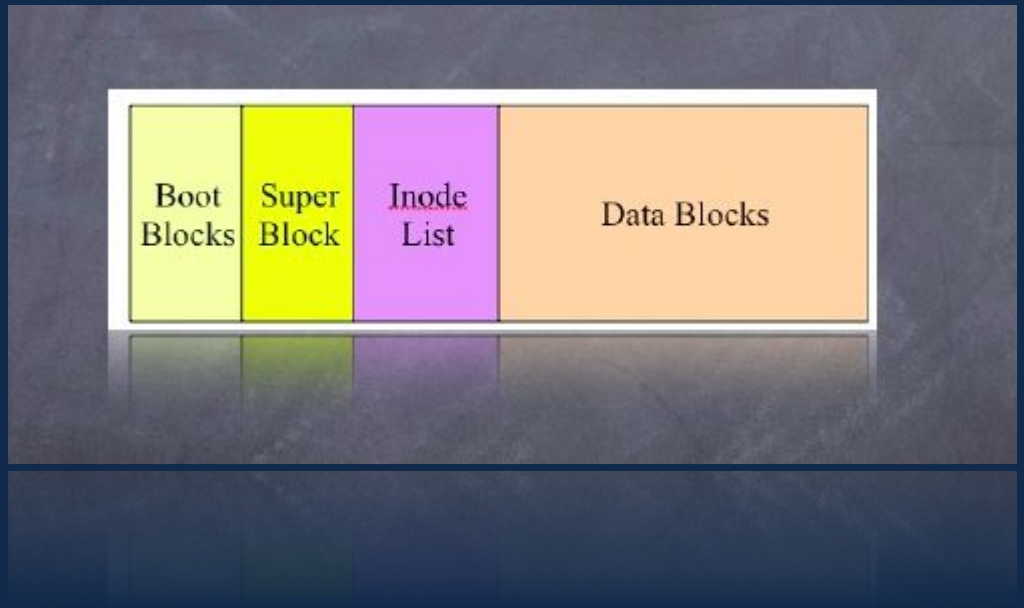
Implementation
Of
Inode based File System

-Team AmigOS

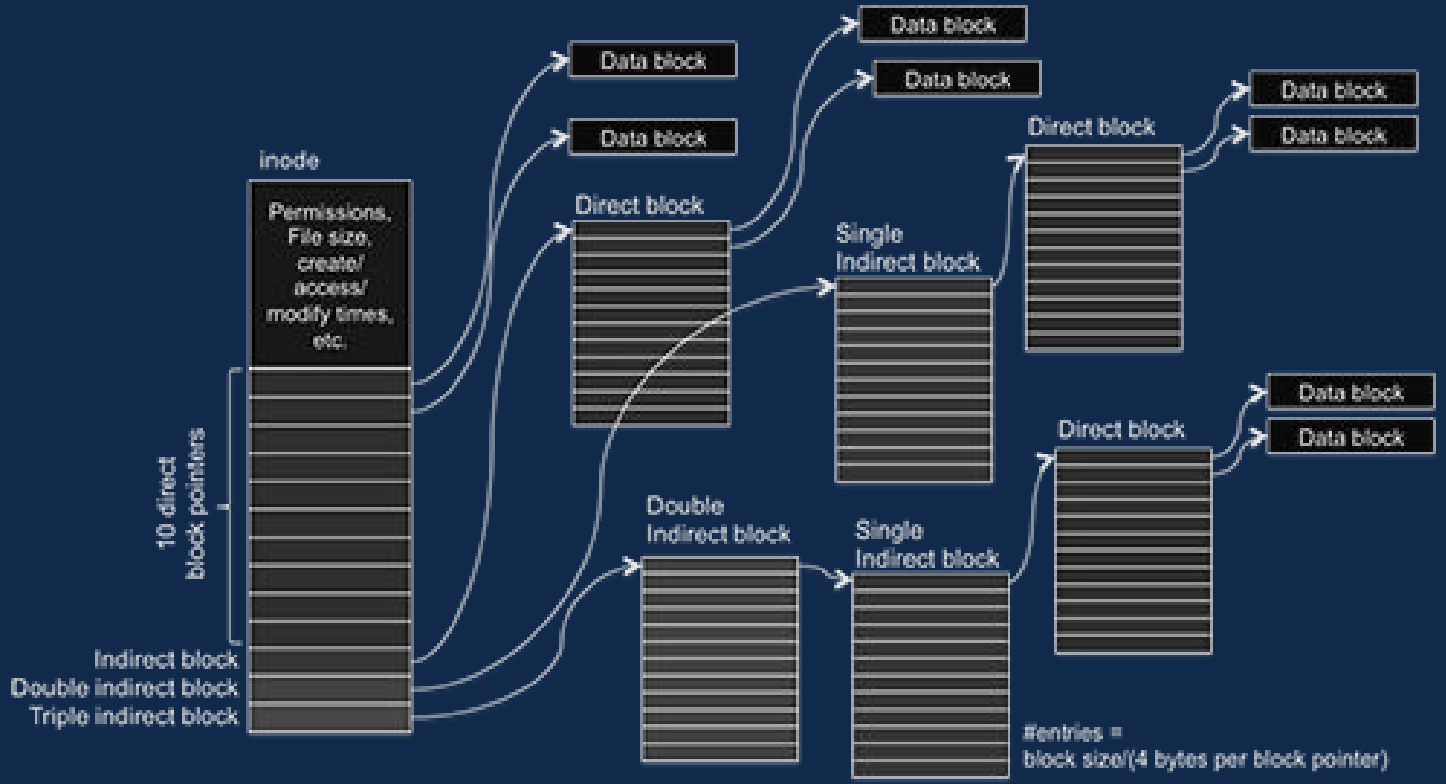
File System Structure



inode structure

```
struct inode
{
    int inode_num;
    int type;
    int filesize;
    int pointer[13];
};
```

Inode structure expanded



Implementation:

- Meta Data structure (Superblock -SB, Directory Structure-DS, inode list)
- Create a virtual disk
- Mount the disk (existing / new created)
- Open existing file
- Read data from file
- Close file
- Create a empty file
- Write into file
- Delete a file from disk
- List all stored files
- Unmount the disk

Create Disk:

1)Create and initialize the Virtual Disk.

2)Initialize SuperBlock
-Initialize the freelist information of
Inodes data blocks & store it in Virtual Disk

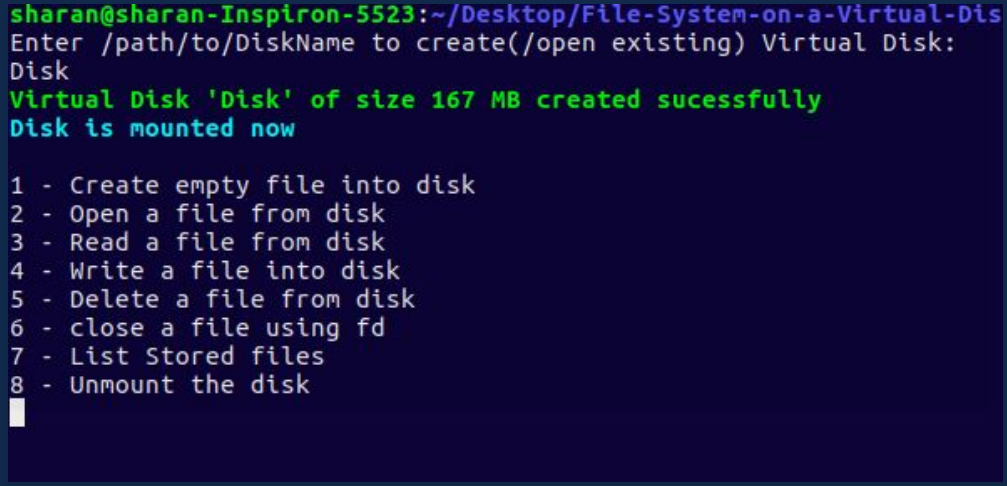
3)initialize the Directory Structure and store it in Virtual Disk.

4)Initialize the Inode list and and store it into Virtual Disk.

Mount File System

1) Retrieve the Metadata of Virtual Disk which includes:
   a) Retrieve Super Block
   b) Retrieve Directory Structure
   c) Retrieve Inode List

2) Display basic information regarding disk.



Open File

1) It gives an error if
   a) File does not exist. OR
   b) No filedescriptor is available.

2) Open the file and assign a filedescriptor to it.

File Read :

1) File with specified filedescriptor should be opened
2) Start reading from the cursor position and store the read data into a buffer
3) Store the buffer content into the new file

Close File

1) File should be opened with specified file descriptor.

2) Then close the file and make the particular file descriptor available.

Create File

1) It will show an error if :
   a) File already exists.
   b) No data block is free
   c) No inode is free.

2) Then it will pick one Data block and one Inode and create the file

3) Make an entry in Directory Structure :
   a) Store the name of file
   b) Store the inode number assigned to that file.

Write data into File(fd, block, buf)

Flow:

- Receive data from application (user)
  Cur_pos, buf, inode etc
- Check disk available space
- Write into last DB (if space remaining)
- Else assign new empty DB to file & write
- Update inode structure (pointer) & file size & set cur_pos
- Return success

```
1 - Create empty file into disk
2 - Open a file from disk
3 - Read a file from disk
4 - Write a file into disk
5 - Delete a file from disk
6 - close a file using fd
7 - List Stored files
8 - Unmount the disk
4
Enter source path/to/filename: images.jpeg
entry created in dir_map for filename = images.jpeg & inode no assigned =0
 images.jpeg file is created successfully.
Given 0 as File descriptor
filesize before writing anything = 0
FD 0 closed successfully
File written Successfully
```

Delete File:

Logical Flow:

1)File should exist.

2)File should not be opened.

2)Empty the data contained in inode:

- Empty the meta data
- Empty the pointer fields

3)The inode and data blocks which are free are kept in freelist.

4)Delete the entry from directory map.

5)On success returns 0 else -1.

List Files

List all the stored files from DS

```
1 - Create empty file into disk
2 - Open a file from disk
3 - Read a file from disk
4 - Write a file into disk
5 - Delete a file from disk
6 - close a file using fd
7 - List Stored files
8 - Unmount the disk
7
1.jpeg   17 KB
2.mp4    42730 KB
3.pdf    84 KB
4.tar.gz   2 KB
```

Unmounting The Disk:

Logical Flow:

1)Store the changed Metadata information about the disk into the disk which involves

=>update and store the Superblock (SB)

=>store the Directory Structure (DS)

=>store the inode list.

Unmounting The Disk:

Store the changed Metadata information about the disk into the disk which involves

Step 1	Step 2	Step 3
Update and store the Superblock (SB)	Store the Directory Structure (DS)	Store the inode list.

Thank You