AHMEDABAD UNIVERSITY | School of Engineering and Applied Science

Operating Systems Lab

File Management System

Group 6

Dharmik Bhayani - 201501010
Jaymeen Kachrola - 201501042
Gaurav Zalariya - 201501125

Table of Content :

Sr. No.	Title	Page No
1	Title Page	1
2	Motivation	3
3	Brief Description	3
4	What is file management system?	3
5	Need for file management system?	3
6	Commonly used file management systems	4
7	File system architecture	4
8	Technical Specifications	6
9	Flowcharts	8
10	Implementations	17
11	Test Results	19
12	References	20

Motivation :

Since the start of our course Operating Systems we started learning virtual memory which lead us to knowing that a process can be bigger than its main memory which grew curiosity in us to know how big can a file be. So we went on searching on internet about the largest file possible? We found that actually the size of the largest file depends on what file management system it is implemented on. And thus it lead us to this project.

Brief Description :

- Our Project aims at creating of a simple file management system and implement some simple projects such as create directory, read, write etc.
- The project focuses on the use of Inode Data Structure for mapping the disk.
- Signal Handling for interrupts like unexpected break (ctrl+c)
- **Threads for faster access of data in the disk.**
- Implementation of log file to keep record of all the events being done.

What is file management system?

A file management system is that set of system software that provides services to users and applications in the use of files. Typically, the only way that a user or application may access files is through the file management system.

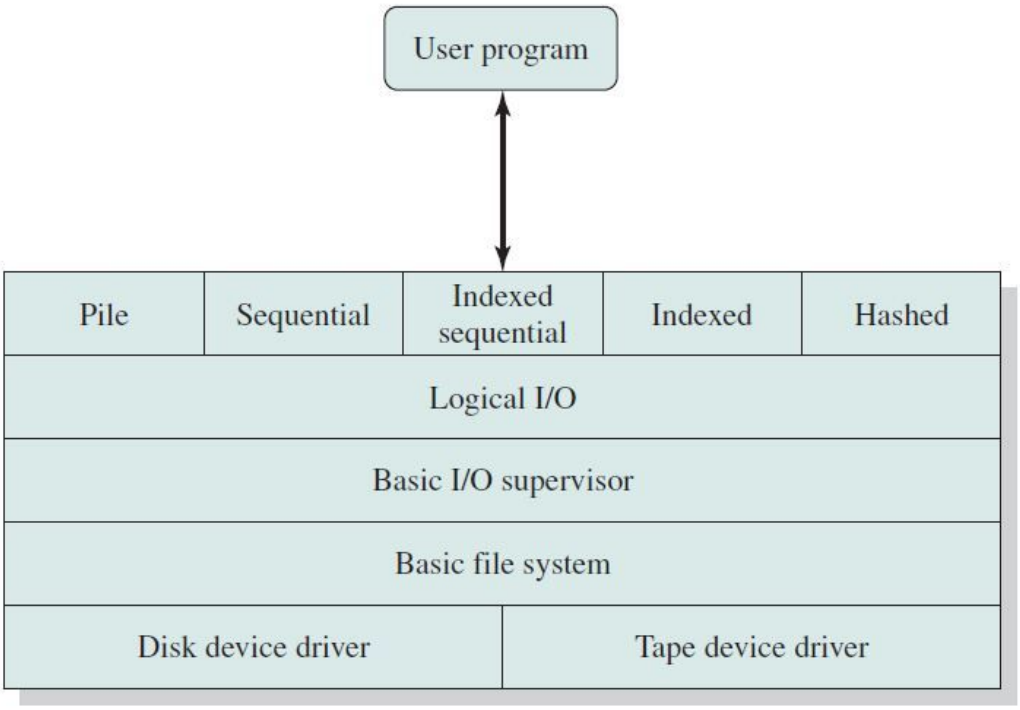
Need for file management system.

The basic objective of a good filing system is to be able to find the record you need quickly and economically, regardless of its format. The goal of a good filing system is to provide quick access to information. Files management is integral part of operating system. It is a logical and practical approach to the creation, maintenance, use and disposition of files and, therefore, to the information that those files contain. It ensures that records are able to be retrieved when needed.

Commonly used File Management Systems.

- NTFS (New Technology File Systems)
- FAT (File Allocation Table)
- ext2, ext3, ext4 (Extended Filesystems)

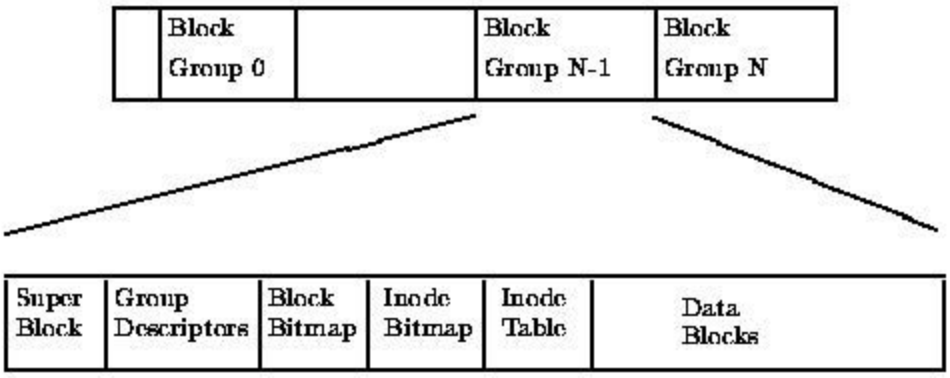
Architecture of File Management System



Levels of Architecture :

- Device Driver level : communicate directly with peripheral devices or their controllers or channels
- Basic File system : Primary interface with the environment outside of the computer system. It deals with blocks of data that are exchanged with disk or tape systems
- Basic I/O : Control structures are maintained that deal with device I/O, scheduling, and file status. The basic I/O supervisor selects the device on which file I/O is to be performed, based on the particular file selected.
- Logical I/O : Logical I/O module deals with file records. Logical I/O provides a general-purpose record I/O capability and maintains basic data about files.

Our project focuses on second extended file system also known as ext2



The physical layout of ext2 is shown above.

The EXT2 file system, like a lot of the file systems, is built on the premise that the data held in files is kept in data blocks. These data blocks are all of the same length and, although that length can vary between different EXT2 file systems the block size of a particular EXT2 file system is set when it is created. Every file's size is rounded up to an integral number of blocks. If the block size is 1024 bytes, then a file of 1025 bytes will occupy two 1024 byte blocks. Unfortunately this means that on average you waste half a block per file. Usually in computing we trade off CPU usage for memory and disk space utilisation. In this case Linux, along with most operating systems, trades off a relatively inefficient disk usage in order to reduce the workload on the CPU. Not all of the blocks in the file system hold data, some must be used to contain the information that describes the structure of the file system. EXT2 defines the file system topology by describing each file in the system with an inode data structure. An inode describes which blocks the data within a file occupies as well as the access rights of the file, the file's modification times and the type of the file. Every file in the EXT2 file system is described by a single inode and each inode has a single unique number identifying it. The inodes for the file system are all kept together in inode tables. EXT2 directories are simply special files (themselves described by inodes) which contain pointers to the inodes of their directory entries.

Technical Specifications :

All the specifications, calculations are done and implemented on a **32 Bit Machine**
Maximum Number of Data blocks = 4096
Size of Each Data Block = 512
Total memory = 512 * 4096 Bytes = 2 MB.
Maximum Number of Inodes = 512
Maximum File name size allowed = 16 Characters

Two types of the files are created

1. Small File - 5120 Bytes
2. Largest File - 70656 Bytes

Structures Implemented :

1. **Inode** :- Entire data structure to store the file information.

Type
Last Access
Created
Owner
Group
Size
Block Count
Direct Block
Indirect Block
Padding

a. Size = 128 Bytes
b. Members
    i. Type: stores the type of inode. Eg: Ordinary register file, directory, symbolic link etc.
    ii. Last Access: The time value of the time at which the file was last accessed.
    iii. Created: The time at which the file was created
    iv. Owner: The owner of the file
    v. Group: Group name under which the file exists. I.e. group of the owner.
    vi. Size: Size of inode
    vii. Block Count: Number of blocks used by inode
    viii. Direct Block: The data blocks directly accessible
    ix. Indirect Block: The first indirect block directing to direct blocks
    x. Padding: It can be used to store some special information or kernel level scheme information and keeps the size of our data set as integral exponent of 2.

2. **Superblock :** Keeps track of the unused inodes and data blocks and is loaded only once in our disk while initial mounting.

Free Block Count
Free Inode Count

a. Free Block Count : Keeps track of the number of free data blocks available.
b. Free Inode Count : Keeps track of the number of free inodes available.

3. **Directory Entry :** Maps inode number with the file name it refers to.

Inode
Name

a. Inode : The inode structure defined above.
b. Name : Name of the file or directory the above inode structure refers to.

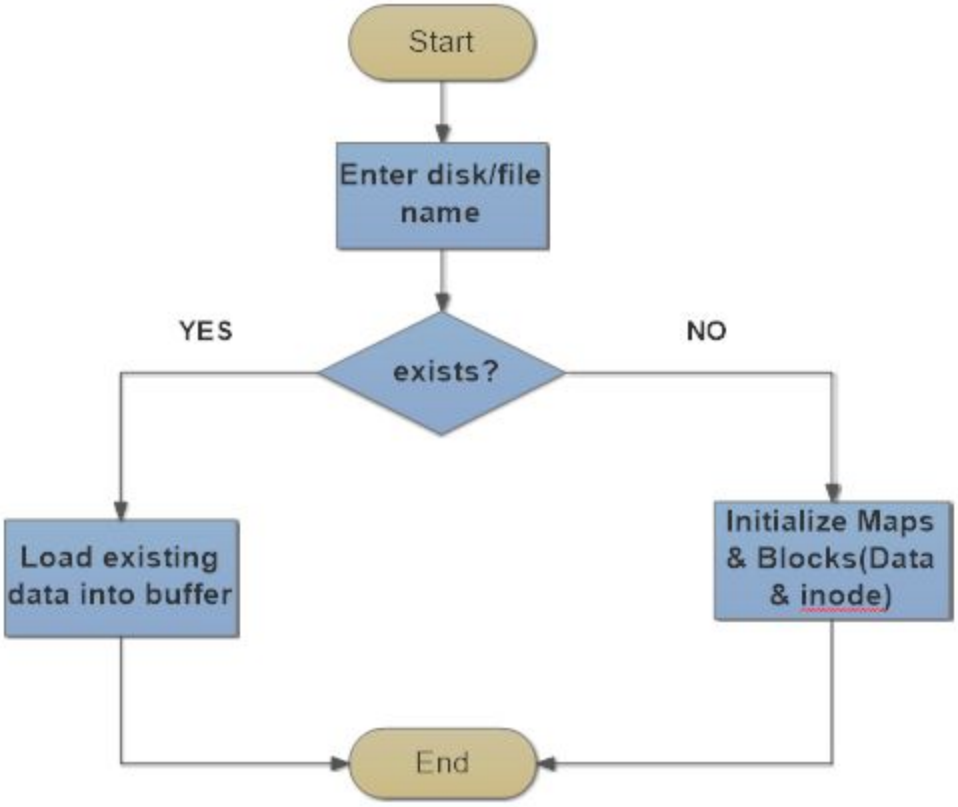
4. **Dentry :** Keeps track of all directory entries in a block.

Directory Entry
NumEntry

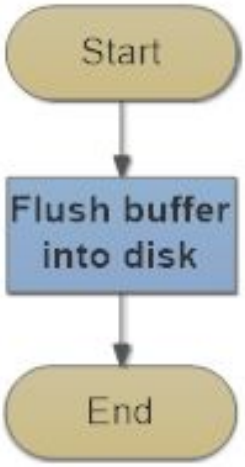
   a. Directory Entry : As defined above
   b. NumEntry : Number of entries in the dentry block

Flow Charts

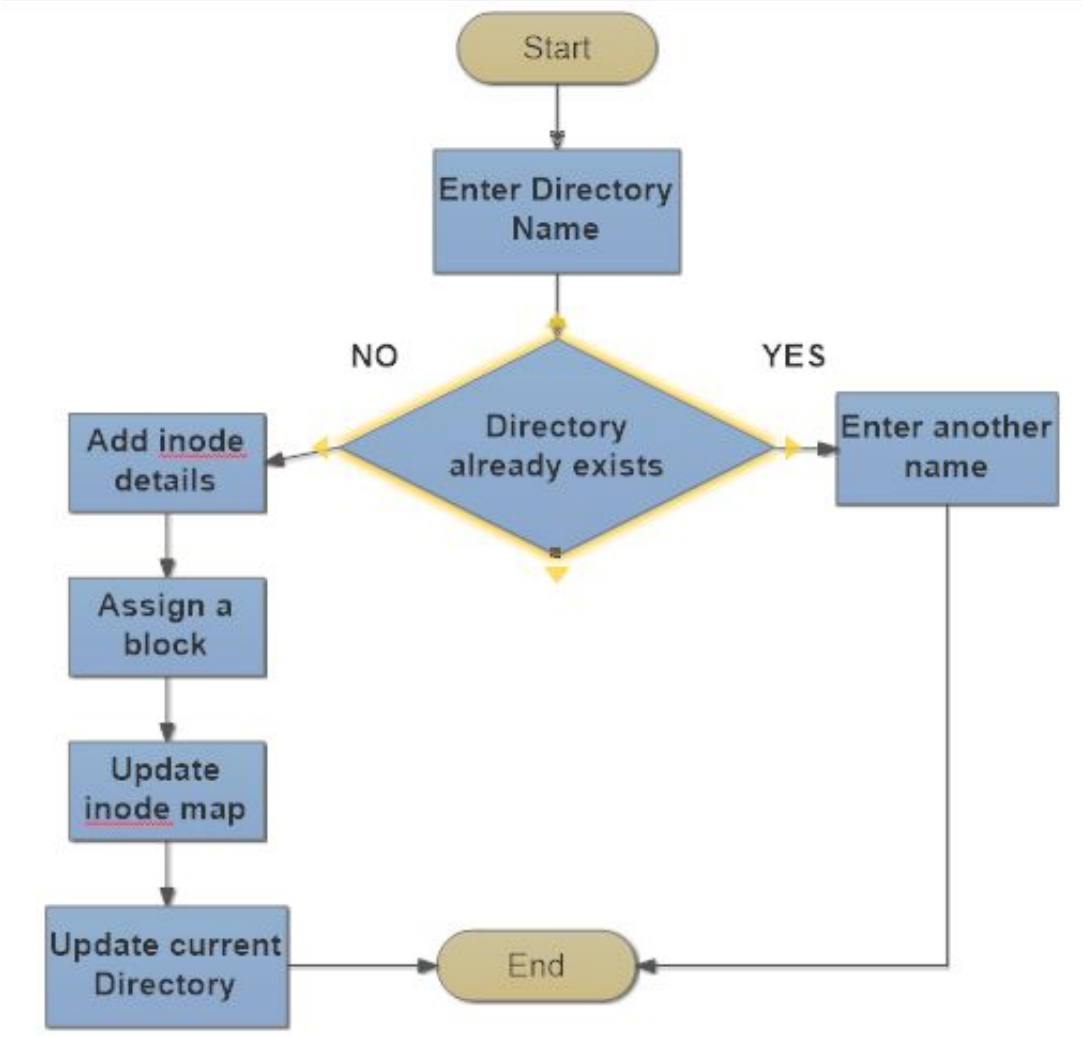
1. Disk Mount



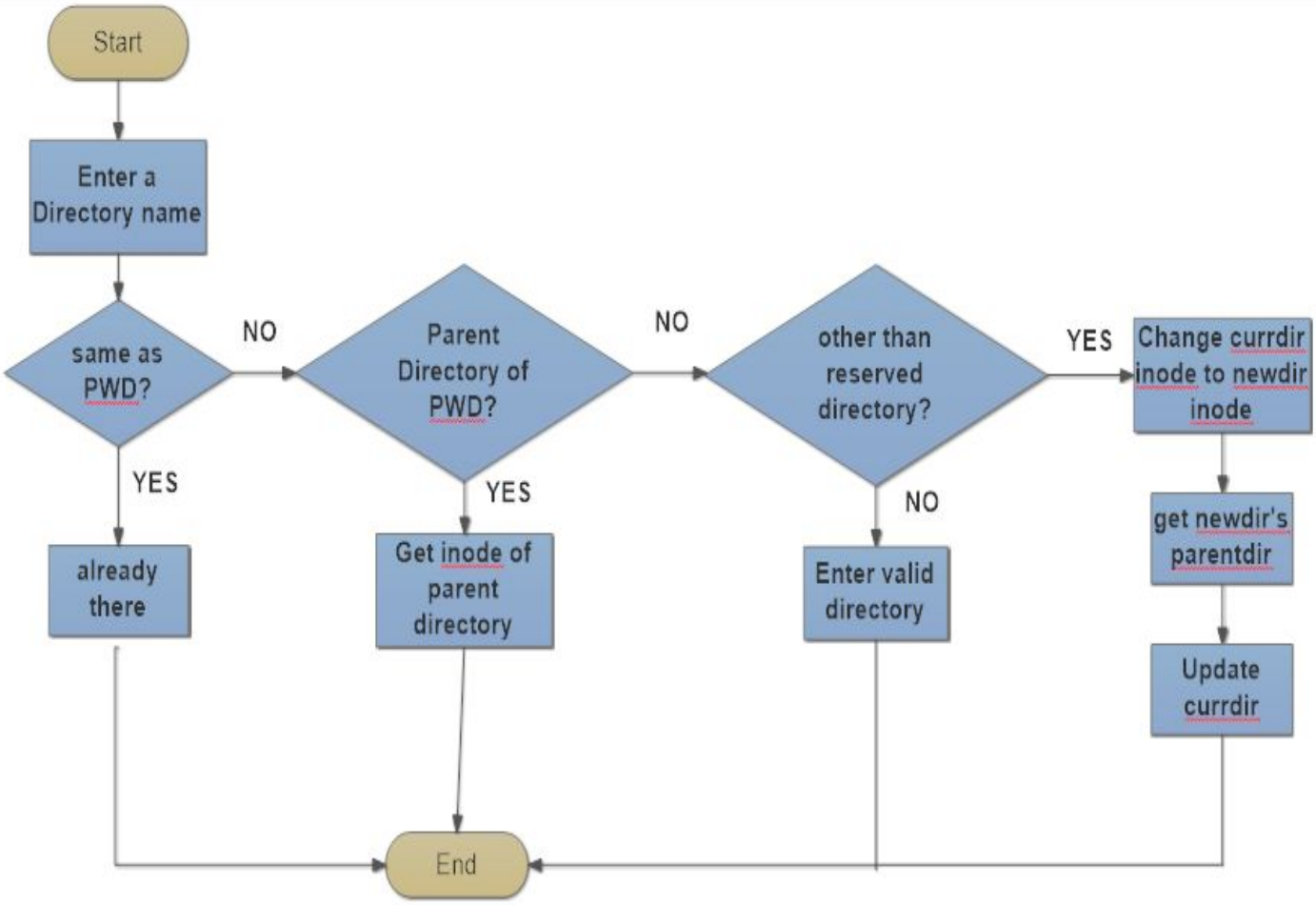
2. Disk Unmount



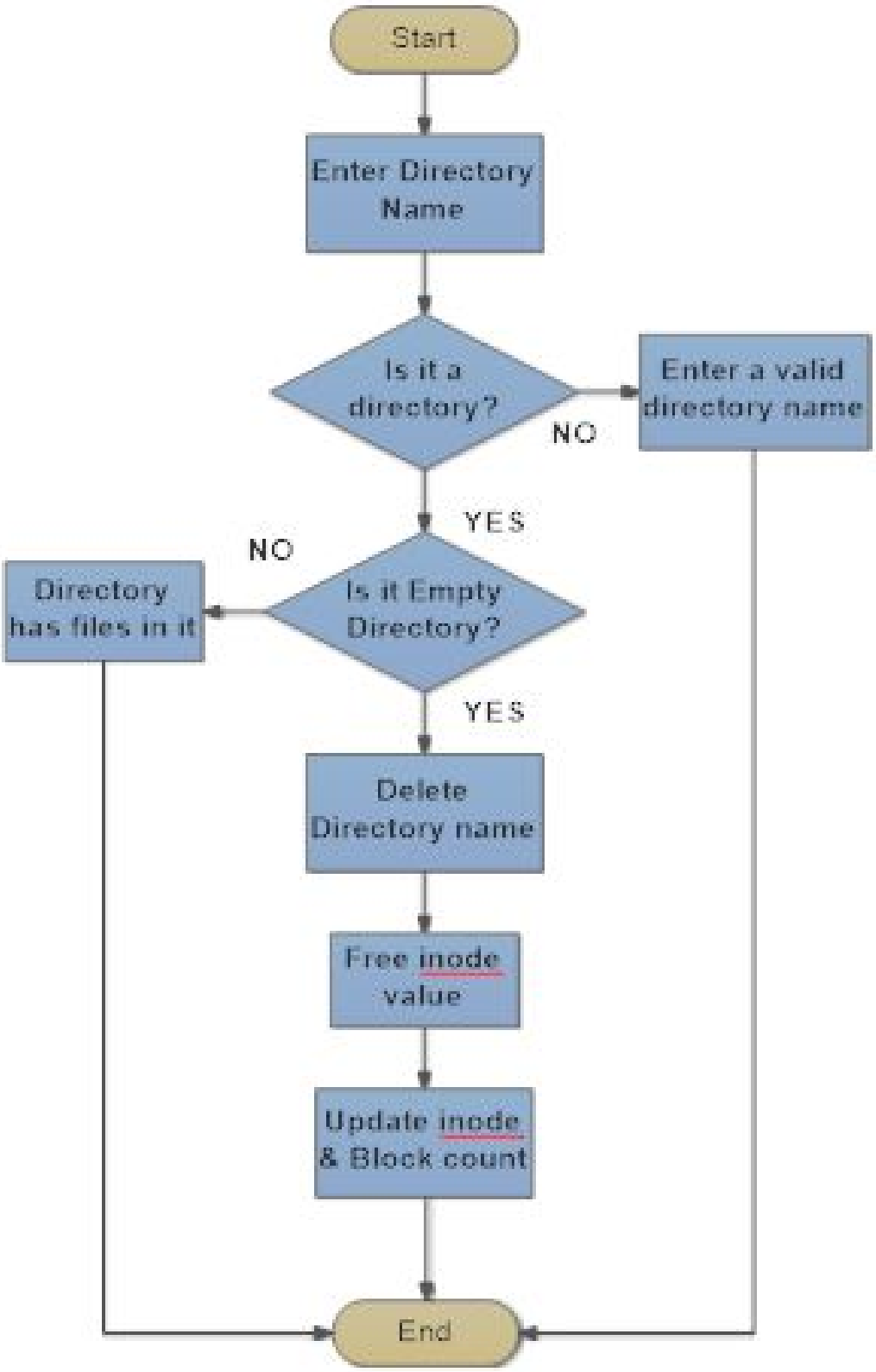
3. Create Directory



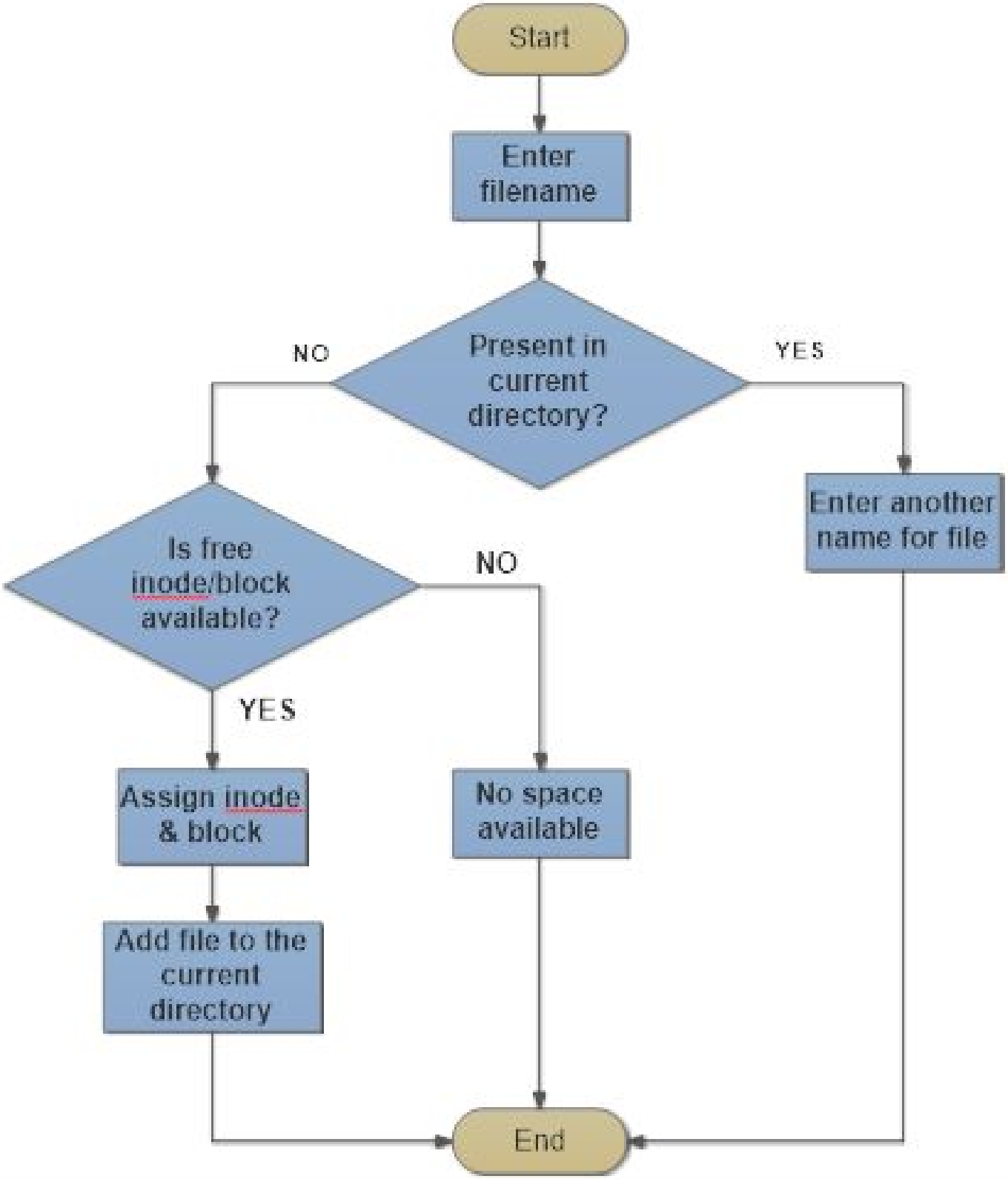
4. Change Directory



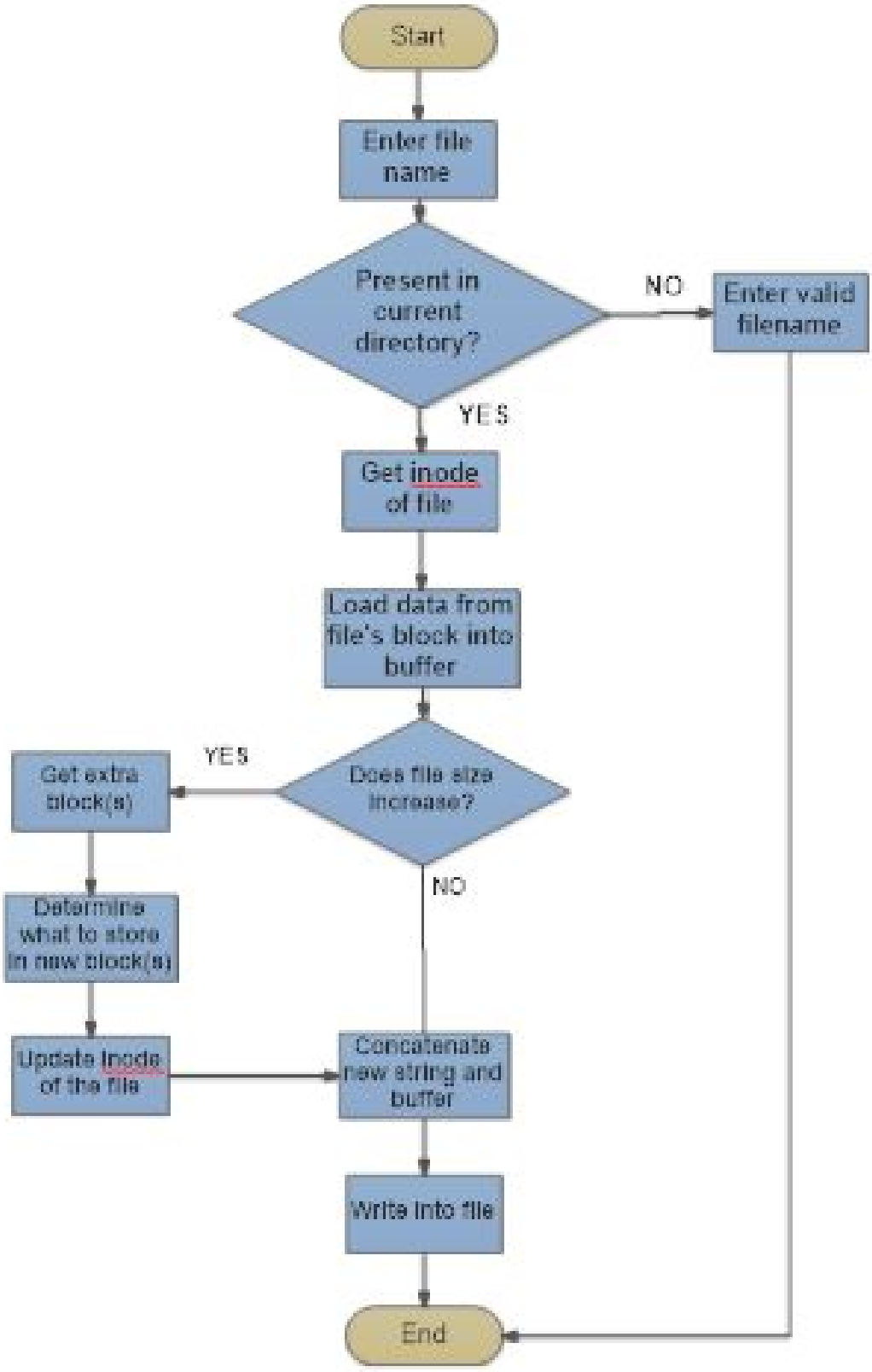
5. Remove Directory



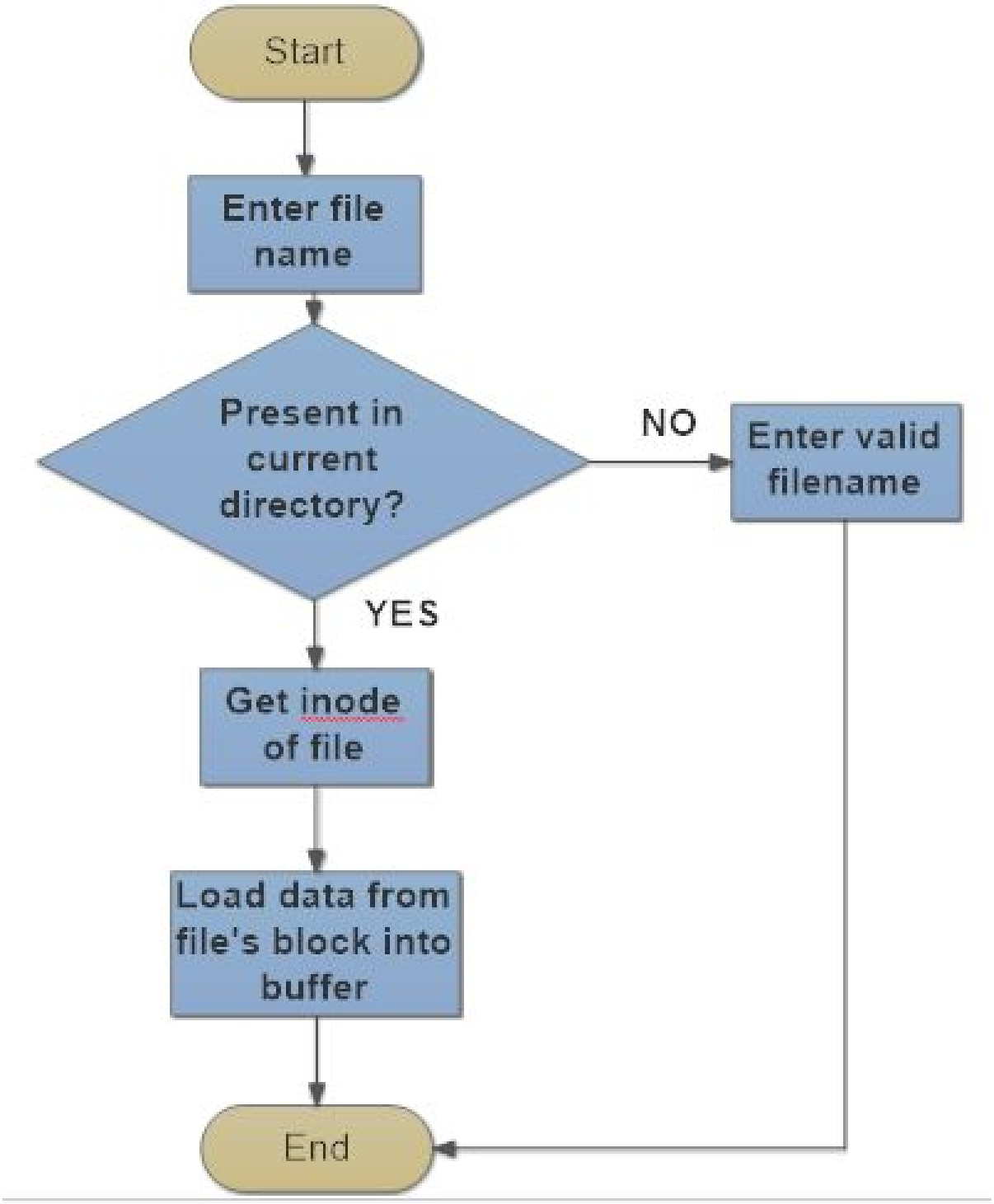
6. Create File



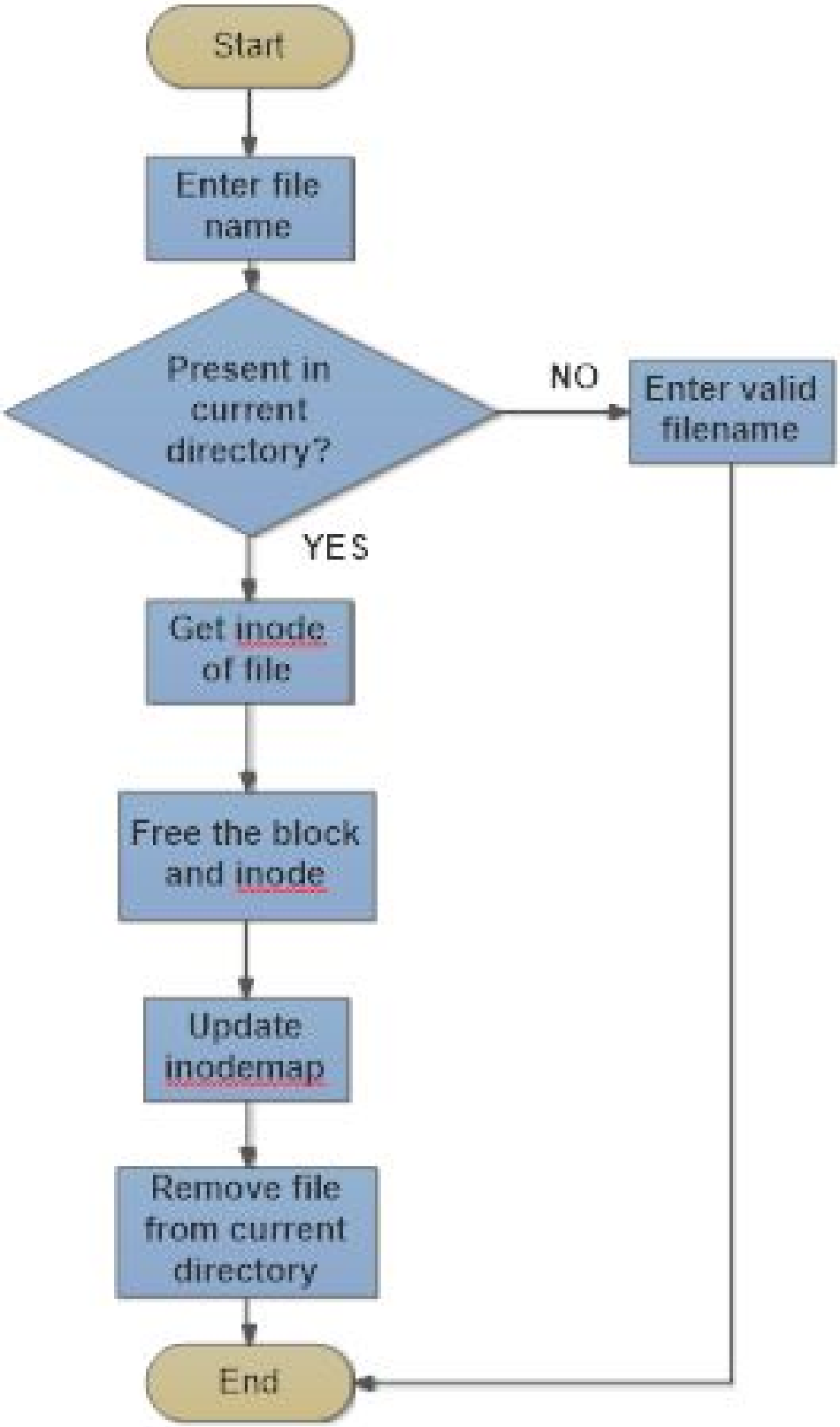
7. Write FIle


Start
Enter file name
Present in current directory?
NO
Enter valid filename
YES
Get inode of file
Load data from file's block into buffer
Does file size increase?
YES
Get extra block(s)
NO
Determine what to store in new block(s)
Update inode of the file
Concatenate new string and buffer
Write into file
End

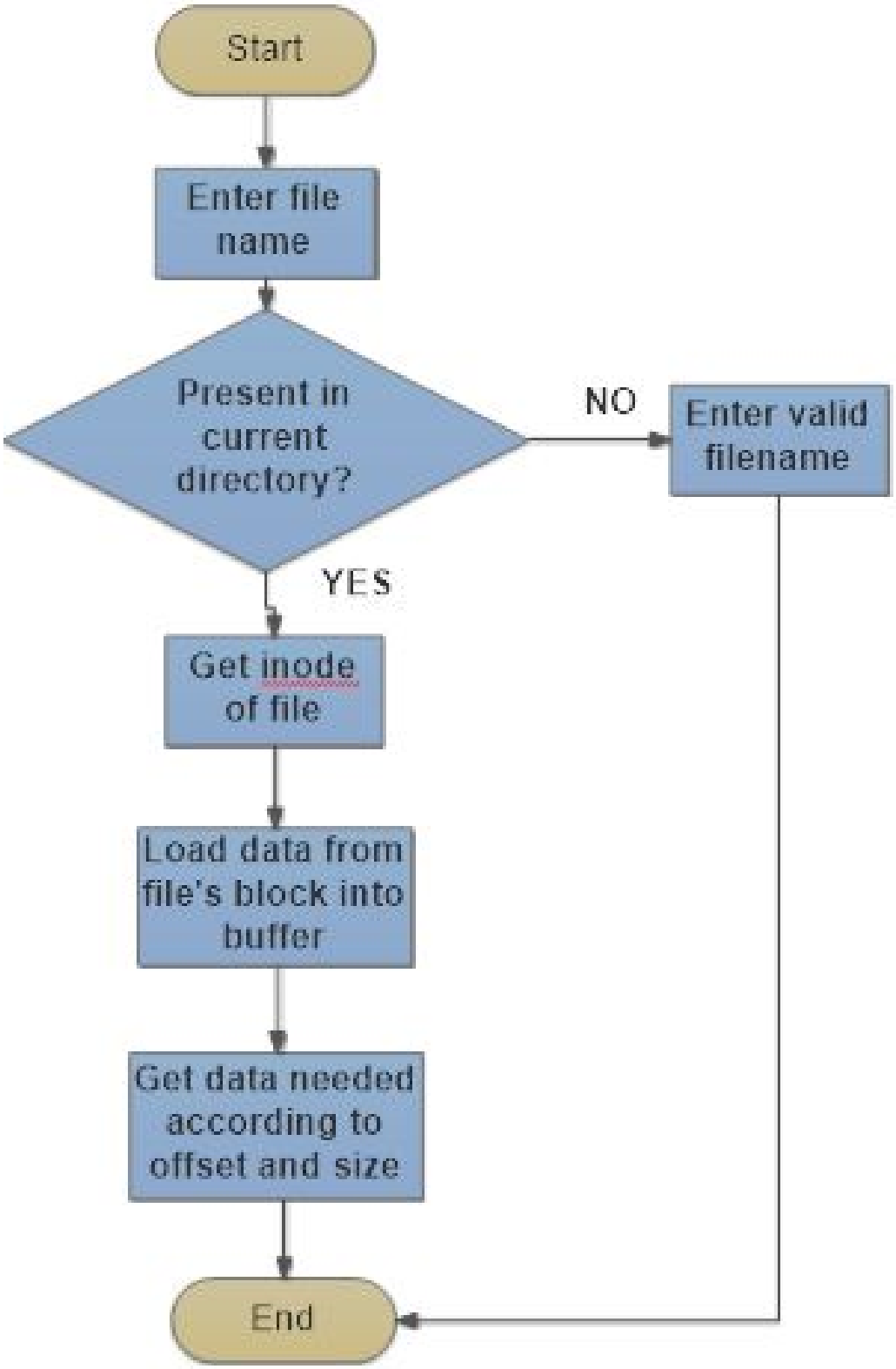
8. Display File



9. Remove File



10. Open file in read mode



Implementation

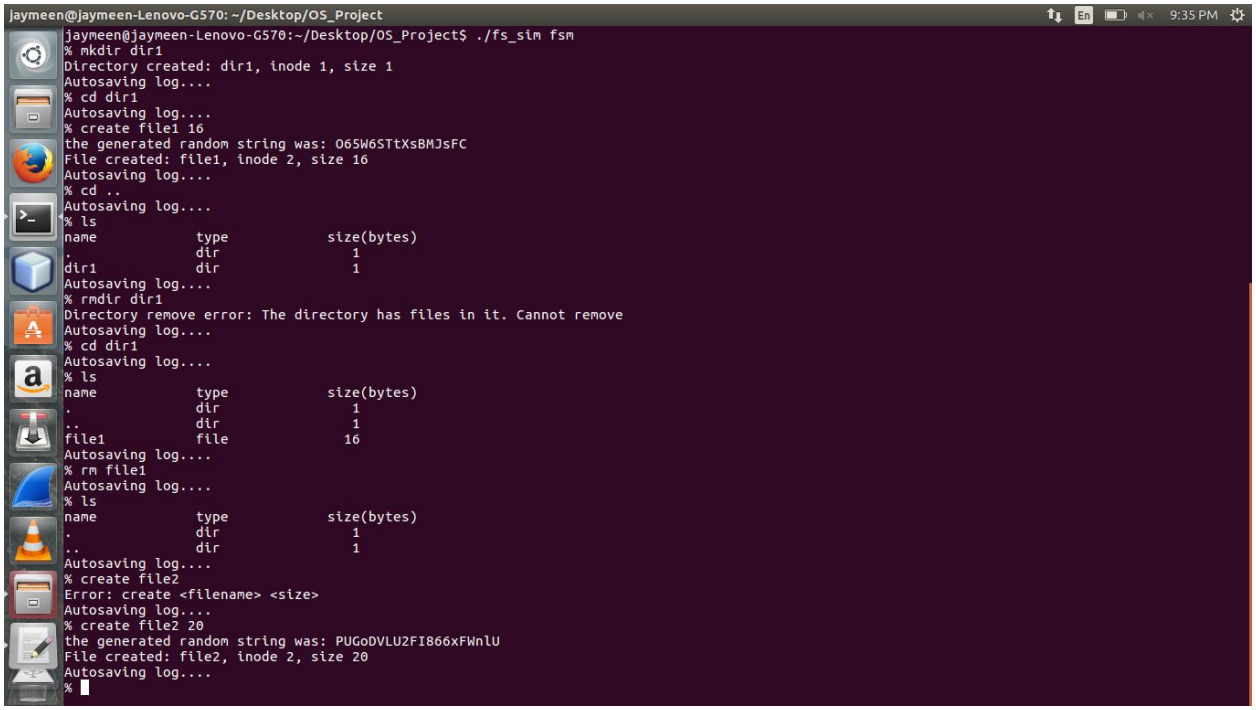
The source code consists of different files such as

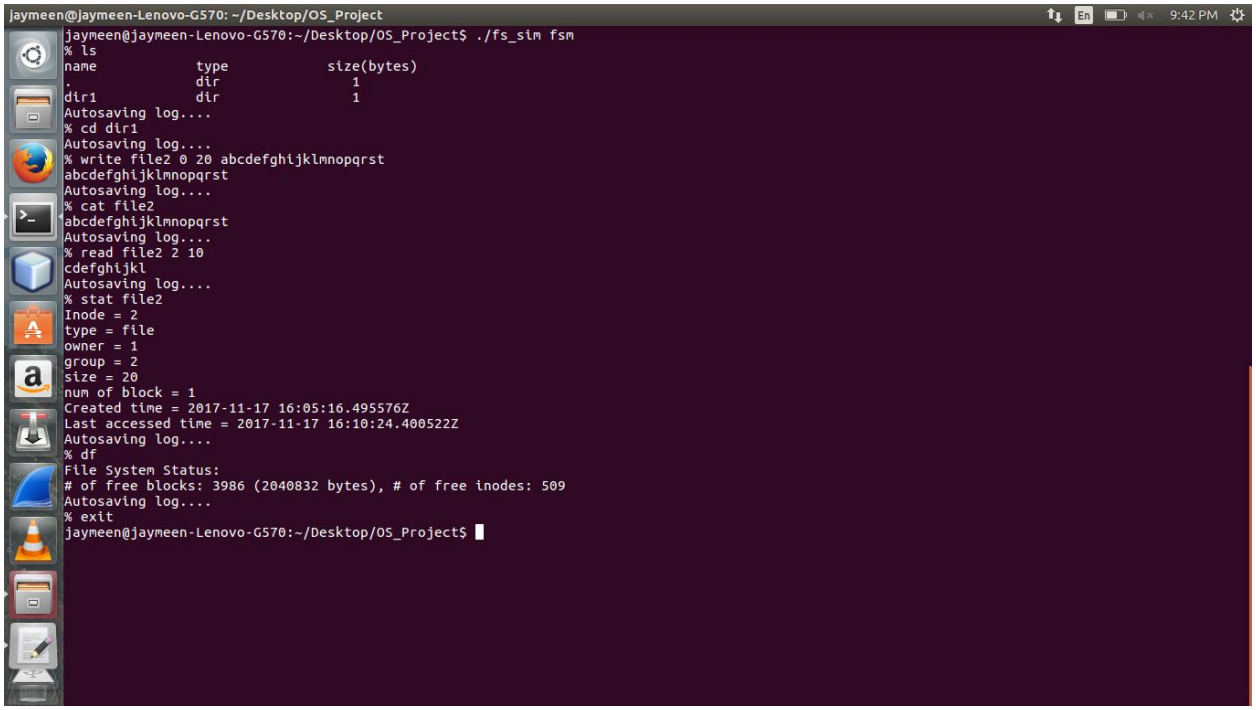
1. Disk.c
    a. Mount disk
    b. Read disk
    c. Write into disk
    d. Unmount disk
2. Fs.c
    a. Managing log file
    b. Mount
    c. Unmount
    d. Search Current directory
    e. Create File (create)
    f. Display File (cat)
    g. Read File (read)
    h. Write into File (write)
    i. Remove File (rm)
    j. File information (stat)
        i. File owner
        ii. File size
        iii. Number of blocks
        iv. Time at which it was created
        v. Last access time
    k. Make Directory (mkdir)
    l. Remove Directory (rmdir)
    m. Change Directory (cd)
    n. List Files in current directory (ls)
    o. Disk Statistics (df)
3. Fs_sim.c
    a. Main Function

4. Fs_util.c
    a. Get random string
    b. Toggle bit
    c. Get bit
    d. Set bit
    e. Get free inode
    f. Get free block
    g. Current time stamp
5. Disk.h
6. Fs.h
7. Fs_util.h
8. Makefile : make file to compile and run the file system.

Project Code is available at : https://goo.gl/bzbgVN
Source Code of Project is available at : https://github.com/aguilfoyle/Project03_CS4730

Test Results





References

- Operating Systems, Internals and Design Principles, 7th Edition, William Stallings
- Basics of file system management
- Alan Guilfoyle, Project on unix file system
- http://www.nongnu.org/ext2-doc/ext2.html
- https://www.youtube.com/watch?v=PbkgiO0YLxc
- https://www.youtube.com/watch?v=BV0-EPUYuQc