FUSE

FILE SYSTEM

**Group Members:**

DEEPIKA H 01FB16ECS108

LIKITHA R 01FB17ECS717

MANJULA K B 01FB17ECS735

Report

INTRODUCTION

* File System:

A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.

File system controls how data is stored and retrieved. It typically manages operations, such as storage management, file naming, directories/folders, metadata, access rules and privileges.

* Here we use **FUSE** implementation.

#### **The python-fuse module**

First of all, to communicate with the FUSE module from Python you will need to install the **python-fuse** module. This module is just a simple interface to FUSE and MacFUSE. Install it by using apt-get:

apt-get install fuse pytho-fuse

FUSE is composed of two parts - a kernel module and a user space library. Once the module is loaded, an application program can be linked with the userspace library and executed in the background. It is this program which implements the filesystem. As the code is not part of the kernel, it is very easy to do things like network and file I/O and you can think of creating all kinds of interesting `pseudo' file systems like the Gmail drive. If you are a Python fan like me, you can even implement your filesystem as a Python application.

PHASE 1:

FILE SYSTEM DESIGN

* Data Structure used: Trees

class Node(object):

def \_\_init\_\_(self,data):

self.data=data

self.child=[]

def add(self,obj):

self.child.append(obj)

methods:

def insert(node,parent,value): #to insert child into the tree

def remove(node,parent,child): #to remove child from tree

def disp\_child(node,parent): #returns the children of the given parent

def disp(node): #prints the tree i.e all parents and children

The structure and design of the file system is as follows:

* Blocks:

Blocks contain a file’s contents.

* Block size: 512 bytes
* Inode:

Inode is the information node and has an inode number associated with it in the file system it resides. It is a [data structure](https://en.wikipedia.org/wiki/Data_structure)  that describes a [file](https://en.wikipedia.org/wiki/Computer_file) or a [directory](https://en.wikipedia.org/wiki/Directory_(computing)). Each inode stores the attributes that include the metadata and as well as owner and permission data and disk block location of the object's data.

Directories are lists of names assigned to inodes. A directory contains an entry for itself, its parent, and each of its children.

And hence, the no. of links for an empty directory is 2.

* Inode structure-

attr = {

'st\_ino': ino,

'st\_mode': S\_IFDIR | 0o755,

'st\_nlink': 2,

'st\_uid': 500,

'st\_gid': 500,

'st\_atime': now,

'st\_mtime': now,

'st\_ctime': now,

'st\_blk':blk,

'st\_size':0

}

* Superblock:

The superblock holds metadata about the filesystem, like which **inode** is the top-level directory and the type of filesystem used.

* Inode Bitmap

The Inode Bitmap contains the free inodes

* Data Bitmap

The data Bitmap contains the free blocks available

* Inode table

The Inode Table contains the stat structure details along with the path of the file or directory

* Data Blocks

Each block contains the data that is stored in the file

For symbolic links it is the name of the file or directory to which it is pointing to.

In disk.txt file,

Super\_Block\_Inode\_bit\_map\_addr 4

Super\_Block\_Data\_bit\_map\_addr 5

Inode\_addr 6 21

Data\_Block\_addr 22 37

The numbers represents the line number where the actual inodes and data blocks are present.

PHASE 2:

SYSTEM CALLS

* Functions used-
* def mkdir(self, path, mode):

#Create a directory with the given name . The directory permissions are encoded in mode.

* def getattr(self, path, fh=None):

#Return file attributes. This call is pretty much required for a usable filesystem.

* def readdir(self, path, fh):

#Return one or more directory entries.

* def open(self, path, flags):

# Open a file.

* def create(self, path, mode):

#create a file. Allocate any necessary structures

* def write(self, path, data, offset, fh):

# Write data into the file from the specified offset onwards. If offset=0 then it is like truncating the data which was earlier present and writing the data into the file.

* def read(self, path, size, offset, fh):

#Read size bytes from the given file and returns the data that is read from the offset, if offset was at or beyond the end of the file. Required for any sensible filesystem.

* def unlink(self, path):

#Remove (delete) the given file, symbolic link, hard link, or special node.  If you support hard links, unlink only deletes the data when the last hard link is removed.

* def truncate(self, path, length, fh=None):

#Truncate or extend the given file so that it is precisely length bytes long. This call is required for read/write filesystems, because recreating a file will first truncate it.

* def chmod(self, path, mode):

#changes the file or the directory permissions with the given mode

* def chown(self, path, uid, gid):

#changes the user id and group id specified in the argument of the given file or directory

* def rmdir(self, path):

#used to remove the file or directory specified from the filesystem.

* def readlink(self, path):

#to read the symbolic link file

* def symlink(self, target, source):

#to create the symbolic link files with target as thelink file and the source as the file to which the symbolic link is pointing to.

* def rename(self, old, new):

#Renames the old file with new name specified.

* def utimens(self, path, times=None):

#To change the access time and change time associated with the file.

* Helper Functions:

class Memory(LoggingMixIn,Operations):

def \_\_init\_\_(self):

self.files = {}

self.data = defaultdict(bytes)

self.path\_inode={}

The init function initialises the class variable file to an empty dictionary which is then populated from the disk.txt file and upon creating new files and directory

The data dictionary is also initialised to an empty dictionary with the key as the path name nad value as the data that is stored in the data block

The path\_inode dictionary contains the key as path and inode as the value

# [logging](https://docs.python.org/3/library/logging.html" \l "module-logging) — Logging facility for Python

This module defines functions and classes which implement a flexible event logging system for applications and libraries.

The key benefit of having the logging API provided by a standard library module is that all Python modules can participate in logging, so your application log can include your own messages integrated with messages from third-party modules.

class LoggingMixIn:

log = logging.getLogger('fuse.log-mixin')

def \_\_call\_\_(self, op, path, \*args):

self.log.debug('-> %s %s %s', op, path, repr(args))

ret = '[Unhandled Exception]'

PHASE 3:

MAKING FILE SYSTEM PERSISTENCE

* Persistence:

To make the file system persistent, we store the Meta data of the file in a Disk.txt file before unmounting it. When we remount the filesystem, the data is retrieved from the file.

The the directory tree is initialised

* def writeDiskInode(path):

#This function returns the inode number that is currently available in the inode bitmap and removes the returned inode from the disk.txt filesystem

* def writeInodeMetaData(files,ino,path):

#This function writes the stat structure details to the file along with the path name of the file

* def addBackInode(ino,blk):

#Adds back the inode number to the available inodes in the inode bitmap when the file or directory is deleted. The corresponding stat structure details is also removed.

* def getBlock(path):

#Gets the available block number from the data bitmap and remove that block number from the file.

* def writeBlock(files,dataa,path):

#writes the data to the blocks given the block number and the newline character is replaced with ‘$’ before writing into the file

* def remove\_block(path,files):

#Erases the block data which was written earlier.

* def persistent(self):

#This function initialises the file dictionary with path as the key and value as the dictionary of stat structure

The Tree data structure is also initialised in this function when mounted on the directory.

TEST CASES

Test cases 1-15:

Test cases 16-17: