To design an in-memory file system similar to \*nix systems, I use various data structures and algorithms to represent directories, files, users, and permissions.

Data Structures:

• Directory: A directory can contain other directories and files. I can represent it as a class with attributes such as a name, parent directory, a list of sub-directories, and a list of files.

• File: A file contains data and has attributes like a name and its contents. I can represent it as a class with these attributes.

• User: A user has a name and permissions. I can represent it as a class with these attributes.

• Permissions: I can represent permissions as a class with attributes for read, write, and execute access.

Algorithms:

• Access Path: I can traverse the file system from the root directory to the desired directory or file using a path. The path can be represented as a list or string, with each element representing a directory or file name.

• Setting Permissions and Reading Permissions: For each file and directory, I can associate a user with its corresponding permissions. I can then check the user's permissions before allowing access to read, write, or execute operations.

• Creating/Reading/Updating/Deleting a Directory or a File: I can use standard tree traversal algorithms to create, read, update, or delete directories and files in the file system.

class Permissions:

def \_\_init\_\_(self, read=False, write=False, execute=False):

self.read = read

self.write = write

self.execute = execute

class User:

def \_\_init\_\_(self, name, permissions):

self.name = name

self.permissions = permissions

class File:

def \_\_init\_\_(self, name, content, owner, permissions):

self.name = name

self.content = content

self.owner = owner

self.permissions = permissions

class Directory:

def \_\_init\_\_(self, name, parent=None):

self.name = name

self.parent = parent

self.sub\_directories = {}

self.files = {}

def add\_directory(self, directory\_name):

new\_directory = Directory(directory\_name, self)

self.sub\_directories[directory\_name] = new\_directory

return new\_directory

def add\_file(self, file\_name, content, owner, permissions):

new\_file = File(file\_name, content, owner, permissions)

self.files[file\_name] = new\_file

return new\_file

def create\_file\_system():

root\_directory = Directory("/")

user1 = User("user1", Permissions(read=True, write=True, execute=True))

user2 = User("user2", Permissions(read=True, write=False, execute=False))

# Creating a file and setting permissions

root\_directory.add\_file("file1.txt", "This is the content of file 1", user1, Permissions(read=True, write=True))

root\_directory.add\_file("file2.txt", "This is the content of file 2", user2, Permissions(read=True, write=False))

# Creating a sub-directory

sub\_directory = root\_directory.add\_directory("subdir1")

sub\_directory.add\_file("file3.txt", "This is the content of file 3", user1, Permissions(read=True, write=False))

return root\_directory

# Accessing files and directories via an access path

def access\_path(root\_directory, path):

current\_directory = root\_directory

elements = path.strip("/").split("/")

for element in elements:

if element in current\_directory.sub\_directories:

current\_directory = current\_directory.sub\_directories[element]

elif element in current\_directory.files:

file = current\_directory.files[element]

user = User("user1", Permissions(read=True, write=True, execute=True))

if file.owner.name == user.name or file.permissions.read:

print(f"Content of {element}: {file.content}")

else:

print("You don't have permission to read this file.")

else:

print(f"{element} not found.")

if \_\_name\_\_ == "\_\_main\_\_":

file\_system = create\_file\_system()

# Accessing files and directories using access path

access\_path(file\_system, "/file1.txt") # Output: Content of file1.txt: This is the content of file 1

access\_path(file\_system, "/subdir1/file3.txt") # Output: Content of file3.txt: This is the content of file 3

access\_path(file\_system, "/file2.txt") # Output: You don't have permission to read this file.

access\_path(file\_system, "/non\_existent\_file.txt") # Output: non\_existent\_file.txt not found.

This code demonstrates a basic in-memory file system with support for

creating directories, files, setting permissions, and reading file content.

It also allows accessing files and directories via an access path and checks the

user's permissions before providing access to the file content.