**Lab 6**: **File Management**

**Instructions**

* You are allowed to form groups of 2 students per group.
* The code should be developed within the team and any form of plagiarism will automatically result in zero for both the groups involved without any questions asked.
* Each student is responsible to understand the code being submitted under his or her name. Division of work is not explicitly required; however, each group member will be given average viva marks of the group. This implies that if a group member gets zero in viva and another gets full marks then both will score 50%.
* The submission deadline is one-week midnight 28 Oct 2022 (AB). Viva will be held during the lab hours.
* Any questions and comments on the lab must reach the faculty at least one day before the deadline through email or in person meeting. No questions will be answered after the said time.

**Introduction**

The purpose of this lab is for you to build the file management component of a distributed file management system. You will build the structure for file management to provide access to user(s) to create, delete, update, and query files in the system. You are required to build the system in the language of your choice (python or C/C++ preferred) and in the operating system of your choice. You are also free to make the design choices as discussed in the book or a combination thereof as long as the requirements for the system are met.

**Objective**

By the end of this lab you will be able to build a file structure and understand the operations on files and directories.

**Tools/Software Requirement**

* Programming

**Requirements**

* + - 1. Your program must allow the users to apply following actions on the files.

1. **Create(fName)**
2. **Delete(fName)**
3. Mkdir(dirName)
4. chDir(dirName)
5. Move(source\_fName, target\_fName)
6. **Open(fName,mode)**
7. **Close(fName)**
8. **fileObj.Write\_to\_file(text), fileObj.write\_to\_file(int write\_at ,text)**
9. **fileObj.Read\_from\_file(),fileObj.Read\_from\_file(start,size)**
10. fileObj.Move\_within\_file(start,size,target)
11. fileObj.Truncate\_file(maxSize)
12. **Show memory map**
13. The files created should be maintained in some form of structure which allows some form of directories.
14. Create → should create a file entry in your file structure. This may or may not include creating space in your disk.
15. Delete → should remove the file from your file structure. This may or may not mean deleting the actual content of the file.
16. Move → should change the association of a file in the directory structure and must not require physical movement of the content.
17. Open file → should return a file Object and all read, write, move, and truncate should be through this object.
18. Write to a file → should be through a function and should have two modes:
    1. append mode → writes to the end of the file.
    2. write\_at → should write to a specific point in the file. The write\_at may overwrite data at the location specified.
19. Read from a file → should have two modes:
    1. Sequential access → reads from the first word and returns the entire content.
    2. readFrom (start, size) → reads from the start memory location for size number of characters.
20. Move content within a file → this allows the user to move forward or back data. The input should be of the form: Move(filename, from, to, size) where filename is the name of the file, from is the starting location of data to be moved, to is the location where the data should be placed and size is the size of the data to be moved.
21. Truncatefile(size) → should reduce the size of the file to size. Data within the file in memory location after size should be considered deleted.
22. Show memory map → should show the distribution of files in the memory.
23. The system must maintain persistent data.
24. **You are only allowed to create a single data file which should contain all the internal structure and data of the user-created files.**
25. The positive change of the size of the file should be automatic (i.e. if more data is written on a file, or data is moved within a file to a new location then the size should automatically increase).

**Tasks**

Design a program to implement the aforementioned functions and provide an interface to execute the functions.

|  |
| --- |
| **CODE:** |
| import json  import pickle  #file object  class File:  def \_\_init\_\_(self, data : str = "") -> None:  self.data = data  self.mode = ""  self.open = False  def write(self, text : str):  assert(self.open)  if self.mode == 'w':  self.data = text  elif self.mode == 'a':  self.data = self.data + text    def read\_from\_file(self):  assert(self.open)  print(self.data)  def read\_certain\_content(self, start, size):  assert(self.open)  print(self.data[start:(start+size)])  def write\_at\_location(self, location, text : str):  assert(self.open)  self.data = self.data[0:location] + text    def truncate(self, location):  assert(self.open)  self.data = self.data[0 : location]    def close(self):  assert(self.open)  self.open = False  def move\_within\_file(self, start, size, target):  assert(self.open)  move\_chars = self.data[start, start + size]  self.data = self.data[0:start]+self.data[(size+start):target]+move\_chars+self.data[target:len(self.data)]  def get(dictionary : dict,list : list) -> dict:  if len(list) == 1:  return dictionary[list[0]]  else:  return get(dictionary[list[0]],list[1:])  # will filter out all the empty strings after splitting  def get\_keys\_list(text : str):  return filter(None, text.split('/'))  class FileSystem:  def \_\_init\_\_(self) -> None:  self.catalogue = {'/' : {}} # '/' is the root folder  #Creates a file in the directory specified with '/' delimiter. Example create('Hello/Meow/mahad.txt') will create 'mahad.txt' in Hello/Meow nested directory  #All paths must be relative to root  def create(self, name : str) -> None:  keys\_list = [ '/', \*get\_keys\_list(name)] # => [ "/", "dir1", "whattaburger"]  get(self.catalogue, keys\_list[:-1]).update({keys\_list[-1] : File()})  #Creates a directory in the specified path. Each directory node is a python dictionary to accomodate its files  def mkdir(self, name : str) -> None:  keys\_list = [ '/', \*get\_keys\_list(name)]  get(self.catalogue, keys\_list[:-1]).update({keys\_list[-1] : {}})  #delete both file or directory  def delete(self, name : str) -> None:  keys\_list = [ '/', \*get\_keys\_list(name)]  get(self.catalogue, keys\_list[:-1]).pop(keys\_list[-1])  def move(self, name : str, target\_directory):  source\_keys\_list = [ '/', \*get\_keys\_list(name)]  target\_keys\_list = [ '/', \*get\_keys\_list(target\_directory)]  #updating the catalogue  target = get(self.catalogue, source\_keys\_list[:-1]).pop(source\_keys\_list[-1])  get(self.catalogue, target\_keys\_list).update({source\_keys\_list[-1] : target})  def open(self, name : str, mode : str) -> File:  source\_keys\_list = [ '/', \*get\_keys\_list(name)]  f = get(self.catalogue, source\_keys\_list)  f.mode = mode  f.open = True  return f  def close(self, name : str):  source\_keys\_list = [ '/', \*get\_keys\_list(name)]  f = get(self.catalogue, source\_keys\_list)  f.open = False  #displays memory map  def show\_memory\_map():  with open(b"fs.dat", "wb") as f:  pickle.dump(fs, f)  with open("fs.dat", "rb") as f:  FS = pickle.load(f)  print(json.dumps(FS.catalogue, indent=4, sort\_keys=True, default=str))  #generates the object .dat file  def create\_dat\_file():  with open(b"fs.dat", "wb") as f:  pickle.dump(fs, f)  #displays the operations availible in file system  def display\_command\_menu():  print("This file system supports the following commands.")  print("Enter the number associated with each command and the parameters it takes.")  print("\t 1. Create a file\n \  2. Delete a file\n \  3. Create a directory\n \  4. Change the directory of a file or directory\n \  5. Move a file\n \  6. Open a file\n \  7. Close a file\n \  8. Write to a file\n \  9. Read from a file\n \  10. Move within a file\n \  11. Truncate a file\n \  12. Show memory Map.\n \  13. Exit the file system.\n \  Enter your choice: ",end = '')  #handles the menu choices  def handle\_menu(choice):  if (choice == 1):  print("Enter the file path with new name: ",end='')  file\_name = input()  fs.create(file\_name)    elif (choice == 2):  print("Enter the file path to delete: ",end='')  file\_name = input()  fs.delete(file\_name)    elif (choice == 3):  print("Enter the new directory path with new name: ",end='')  dir\_name = input()  fs.mkdir(dir\_name)    elif (choice == 4):  #this command is redundant in this file system  pass    elif (choice == 5):  print("Enter the path of file or dir to move: ",end='')  file\_name = input()  print("Enter the destination path: ", end='')  dest\_path = input()  fs.move(file\_name, dest\_path)  elif (choice == 6):  print("Enter the path of file to open: ",end='')  file\_name = input()  print("Open the mode you want to open the file in. You can type 'w' or 'a': ",end='')  mode = input()  global File  File= fs.open(file\_name,mode)  elif (choice == 7):  print("Enter the path of file to close: ",end='')  File.close()  elif (choice == 8):  print("Do you want to write in default mode or specific location? Enter 1 or default, 2 for a location: ",end='')  write\_method = int(input())  print("Enter the content you wish to write: ", end='')  write\_string = input()  if (write\_method == 1):  File.write(write\_string)  else:  print("Enter the start location: ",end='')  location = int(input())  File.write(location, write\_string)  elif (choice == 9):  print("Do you want to read in default mode or specific location? Enter 1 or default, 2 for a location: ",end='')  write\_method = int(input())  if (write\_method == 1):  File.read()  else:  print("Enter the start location: ",end='')  location = int(input())  print("Enter the size you wish to read:",end='')  size = int(input())  File.read\_certain\_content(location, size)  elif (choice == 10):  print("Enter start location: ",end='')  start = int(input())  print("Enter the size to move: ",end='')  size = int(input())  print("Enter the target location: ",end='')  target = int(input())  File.move\_within\_file(start,size,target)  elif (choice == 11):  File.truncate()  print("File successfully truncated.")  elif (choice == 12):  print("Displaying memory map...")  show\_memory\_map()  elif (choice == 13):  with open(b"fs.dat", "wb") as f:  pickle.dump(fs, f)  global loop\_var  loop\_var = False  #a sample file system creation with some operations  def sampleFileSystem():  fs = FileSystem()  fs.create("Siraj.txt")  fs.create("Hello.txt")  fs.mkdir("dir1")  fs.create("dir1/shahab.txt")  fs.mkdir("dir1/meow")  fs.create("dir1/meow/hello.txt")  fs.delete("dir1/meow")  file = fs.open("Siraj.txt","w")  file.write("hello world")  file.read\_from\_file()  with open(b"fs.dat", "wb") as f:  pickle.dump(fs, f)  with open("fs.dat", "rb") as f:  FS = pickle.load(f)  print(json.dumps(FS.catalogue, indent=4, sort\_keys=True, default=str))    #sampleFileSystem()  main execution menu  fs = FileSystem()  loop\_var = True  while (loop\_var):  display\_command\_menu()  choice = int(input())  handle\_menu(choice)  print("File System Successfully exited.") |
| **OUTPUT:** |
|  |
| **Sample File System Structure:** |
| There is a function sampleFileSystem() that demonstrates the code, with some basic options. It can be called to viewed and analysed a demo for the file system. Below is the output for sample file structure. |
|  |

**Deliverables**

1. The complete code
2. A sample data file (sample.dat) consisting of files and directories to show your output.
3. A user guide on how to use your system. The user guide must include a description of your directory structure.
4. A small document describing your system design.
5. List all references used for implementing your work.