|  |
| --- |
| FILE MANAGEMENT SYSTEM  User Guide |
| **Class:** BESE-11A  **Submitted By:**  Syeda Fatima Shahid (337346)  Hafsa Malik (341303) |

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

## INDEX

## INTRODUCTION

## Description of File Management System

## Description of Directory System

## USAGE EXPLANATION

## Creating a File

## Deleting a File

## Making Directory

## Changing Directory

## Move File

## Open File

## Read File

## Write File

## Move Within File

## Truncate File

## Show Memory Map

## Exit System

## INTRODUCTION

## Description of File Management System:

## In this lab we have developed the file management component of a distributed file management system in python using nested dictionaries structure. This structure deals with handling files and directories. Files are organized insides directories. Directories can also contain further subdirectories. The management structure provides ways to

## Create directories

## Create files and directories inside root directory

## Access the files

## Update and query files in multiple mode (read, write, reposition, truncate etc)

## Move the files

## Delete the files

## Change the directory

## Get Memory Map

## Description of Directory Structure:

## We have initialized our directory structure with a nested dictionary. This dictionary initially contains an empty root directory. Any further files and directories created are located inside this directory.

directory\_structure = {'root':{}}

Directory and file names are used as key. Each new file object is added as a new key value pair using file name as key and file object as value inside the current directory dictionary.

current\_directory[filename]=file\_obj.to\_json()

Each new directory inside root directory is created using directory name as key and for the value an empty dictionary is initialized. This empty dictionary will get populated as key value pairs of filename: file object on requirement.

current\_directory[dirname]={}

We can summarize the overall directory structure as:

**{‘root’ : { ‘f1name’ : file1\_obj , ‘dirName’ : { ‘fname’ : f\_obj } , ‘f2name’ : file2\_obj , ……}}**

## USAGE EXPLANATION

## Creating a File:

## To create a file, we simple select option 1 from the menu. This gives us further three options:

## Enter file name

## Enter file extension

## Enter file content

## Enter the filename of your choice and the type of file you want in extension for example for test file write txt. Finally add the content and enter.

## 

## Deleting a File

## To delete a file, select option 2 and give the absolute path to the file. For example, if the file is located in root folder the absolute path would be ‘root/filename’. This will first split the absolute path based on / delimiter, delete last element and return updated directory structure.

## 

## Creating Directory

## To create a directory, use option 3 and enter directory name as input. This will create a directory inside current directory. The new directory will contain a ‘prev’ key that shows that it is a subdirectory, and it will be used in case user wants to go back to parent directory.

## Text Description automatically generated

## Changing Directory

## To change directory, select option 4 and enter the name of the directory you want to move to. This will take the user entered directory name and give it along with current directory as parameter to traverse() function and output the updated current directory.

## 

## Move File

## To move a file from current directory to another directory use option 5 and enter source file path and target location.

## 

## Open File

## To open a file use option 6 and enter filename , and the file mode you want to open it in. This will return a file object and further action will be performed based on whatever mode user chose. There are 6 mode option each of which is covered below.

## 

## Read Mode (r)

## To read all the content of file use this option.

## 

## Read From Mode (rf)

## If user wants to read only a particular section of file use rf mode and enter start position and end position (index) of file.

## Text Description automatically generated

## Write Mode (w)

## To add new lines and data at the end of file use this mode by giving the content you want to write as user input.

## 

## Write At Mode (wa)

## To write at a specified point in file use wa mode and give that position as input. Let’s say I want to write in file2.txt at index 4 instead of at the end of file I will give the content I want to write and where I want to write (4) as input.

## Text Description automatically generated

## Move Within File (mv\_in)

## To reposition content within file use mv\_in mode. This allows the user to move forward or back data. Give the starting location of data to be moved, end location where the data should be placed and the size of data to be moved.

## Text Description automatically generated

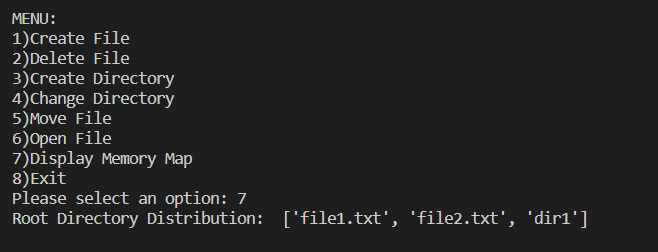
## Truncate Mode (t)

## To reduce the size of the file use this mode. Data within the file in memory location after size will be considered deleted. Give the index till which you want to keep data as input. For example if I want to keep data till index 10 in file2.txt I will give 10 as input.

## 

## Show Memory Map

If the user wants to know the distribution of files in directory, simply select the option 7.

****

## Open File

Finally, to exit system select option 8. Any invalid input (not in between 1 to 8) will result in error message and an option to try again till valid option is selected.

Text

Description automatically generated