# Department of Computing

**CS 330: Operating Systems**

**BESE-9A**

**Assignment #01**

**Threaded File Management System**

**Date: 26 Dec 2020**

**Instructor: Mr. Taufeeq Ur Rehman**

**Group Members:**

**1. Hammad Ahmed 266578**

**2. Usama Ejaz 247644**

**3. Muhammad Ateeb 251414**

**File Management System**

**a) System Design:**

We are using an hierarchical directory structure. the information about the directory structure is stored in a tree made-up of nested dictionaries. the root of the tree is the root directory, each non-leaf node of the tree is a directory. the parent of a node is it's parent directory and the children of a node are it's child directories. each node also contains pointers to it's parent directory and to the root directory for quick navigation. the parent pointer of the root directory points to a 'None' object. The leaves of the tree are either empty directories (directories with no children) or files. These 'files' in the tree are actually list objects containing pointers to the actual data of that file.

**b) Choice of Data Structure:**

We have decided to use trees, implemented with python dictionaries, for storing directory structure because of it’s hierarchical nature.

**b) Memory Map:**

The first 1kB of the .dat file is dedicated to storing the directory structure and pointers to 'holes'. holes are the places on the file (or SSD) where deallacoted data is present. (data that once was part of a file, now it's not). after the first 1kB the actual data of the files is stored. while writing to a file the data is first written on available 'holes' and then at the end of file. after the data is written the pointers to this data is stored in the corresponding file list in the directory tree. while deleting a file the actual data is not deleted. simply the pointers of the files data are moved to the 'holes' list and the file list is removed from the directory tree.

**d) System Functions:**

As discussed earlier the program uses a binary data file sample.dat stored in the current working directory of generating and maintaining the directory structure and file data across multiple runs. When the user starts the program simply by calling python main.py from the terminal, the program reads in the sample.dat file as an initialization step and creates an object named SSD to reference it for reading and writing. The design is such that the first 1kb of the the file is reserved to store the directory structure and any operation related to the manipulation of the file system structure is handled by manipulation in this data. The program reads in the first kilobyte of the data and desearilzes it to recover the stored state of the file system. The is metadata holds pointers to the location of the relevant data of each individual file by which data enclosed in an file in our file system is recovered.

Screenshots have been attached in this document of a test run to guide users on how to get started.

Following are major functions implemented in our files ystem program that explain the features of our program in detail:

* **Initialization Code:**

All the code require for the initialization of the resources required by the program such as global variable declarations, arrays and dictionary structure declarations, data file reading objects, initial root directory etc.

* **Save Function:**

the save() function encloses the code required to write the data in memory after serialization to the sample.dat file so that the file system state could be maintained in a consistent manner.

* **Debugging Function:**

the dump() function is utilized for debugging purposes and extracts the data from the sample.dat file for developer to check for unexpected error in the file system state.

* **File System Feature Related Functions:**

These functions are named after their functionalities and represent the various features our file system lets users to perform. The implementation of our file system is such that it resembles in name and functionality the commands and syntax of the LINUX file system:

* + quit() to gracefully exit the file system storing the state of the file system before closing.
  + chdir() to change directory
  + create() to create files
  + mkdir() to create directories
  + move() to perform move operation
  + delete() to perform delete operation
  + tree() to print the working directory tree
  + read() to read data from the file in file system using File class method
  + write() write data to file in our file system using File class method
  + split() to split path into directory and file name
  + mem\_map() to generate a memory map of file system
  + print\_dir() to print the directory path
  + help() to display usage help
  + File class contains utility functions that perform the actual read and write operations on the sample.dat file after all checks and validations are performed by other file manipulation function described above.

Note:

The recommended python version is python3.8 and above.

Libraries used:

os , pickle, re

**d) Limitations:**

**Bellow listed are some of the limitations of our file management system.**

* **Due to the time constraints, the program is currently designed with the notion that the sample.dat file will be present in the current working directory with proper naming scheme.**
* **The program can only serve one user at a time**
* **the metadata storage size is currently hard-coded to 1kb.**
* **File type checks in the file system is not supported yet.**

**e) Screenshots:**

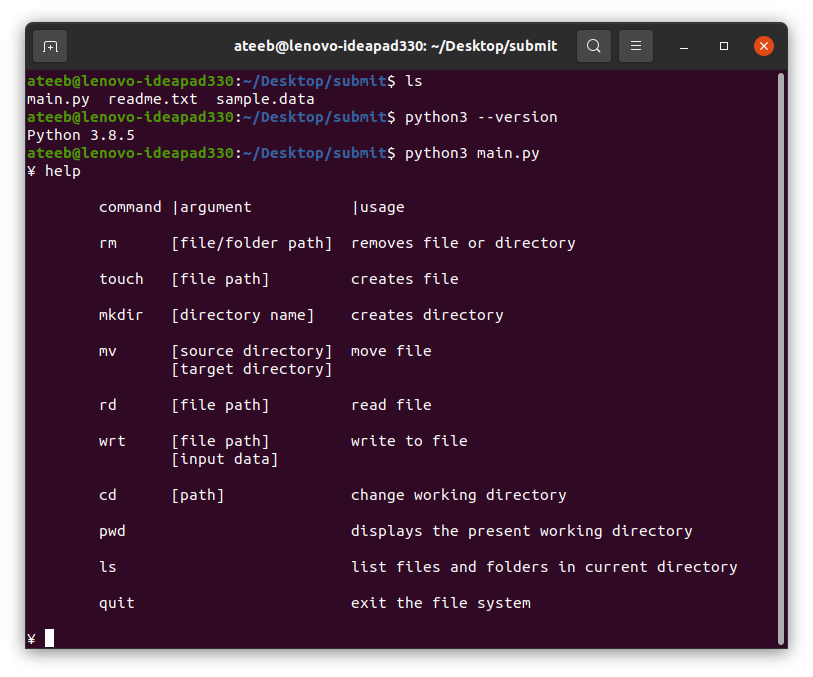
The the screenshots are attached from a test run on python 3.8.5.

Getting Started:

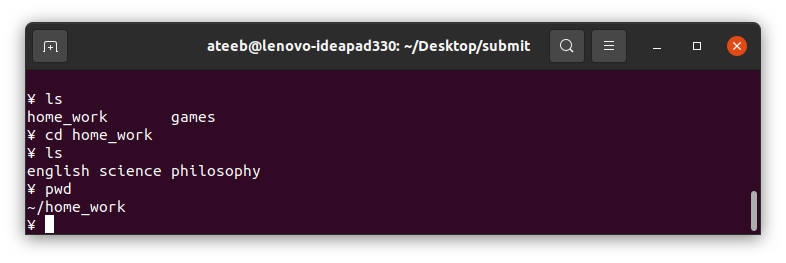
Extract the submitted zip file containing the code and sample data file.

Open terminal in the directory containing the extracted files.

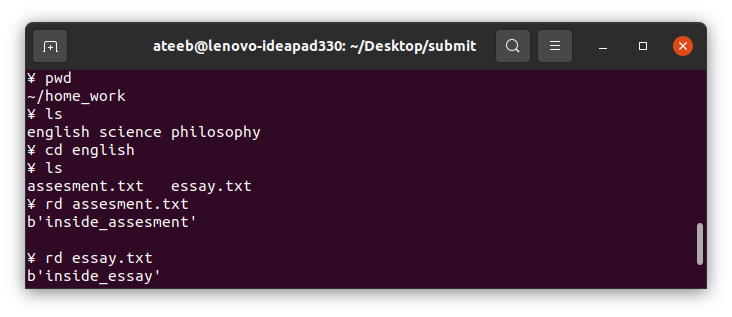
Follow Along the Steps Performed in the Screenshots attached:



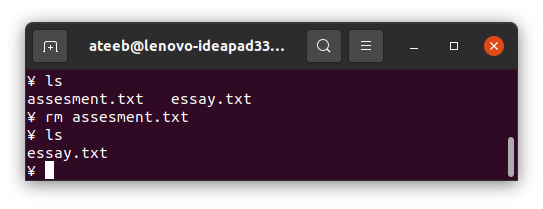
listing the directory | changing directory | returning current directory path



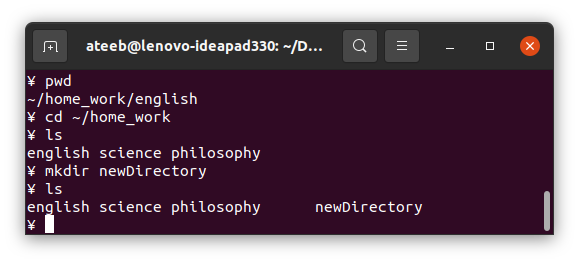
reading files using rd command

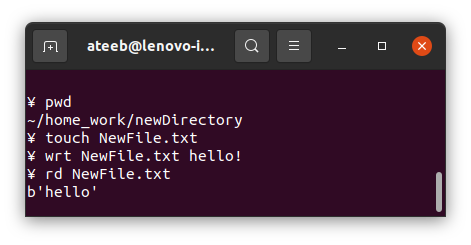


Deleting file using rm



making directory using mkdir

creating and writing to new file



moving file from one current directory

