# Department of Computing

**CS 330: Operating Systems**

**BESE-9A**

**Assignment #02**

**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**

**Threaded File Management System**

**a) Design Modifications:**

For the Design of Our threaded file system we have modified and enhanced the design of our previous single threaded file management system. Below are some of the major modifications implemented to the previous design:

* In order to allow multiple users to use the program simultaneously we have made use of the threading library.
* The program now provides multi-user functionality by the use of a nested python functions. Each incoming user session is handled as separate thread. Each thread calls the threaded function which further calls the required nested function.
* The program is now modified to allow user to the enter the number of concurrent users as an integer argument at run time.
* As a synchronization primitive we have used locks available in the threading library so that concurrent access to the sample.dat file holding the file systems sensitive information could be made possible.
* The writing procedure of the data and it’s metadata to the sample.dat file (shared resource) is now enclosed within locks so that data consistency is maintained.
* An of the core functionalities that were part of the main method earlier have now been moved inside the threaded function so that thread data localization is achieved and we have to deal with minimum number of shared resources.
* The interactive console implementation code is removed as the program now reads user instructions from input files that are generated and saved before-hand.
* Modification has been made so that the program now generates an output file per user input file containing all the results from the performed operations requested by that user. This was each user can check the results of his file system operation by access his output file and avoiding the cluttering of the console.
* A method has been defined to initialize all global resources with appropriate data before the threads are created to utilize these resources.

**b) System Functions:**

The multi-threaded version of our file management system still supports all the functionality supported by the earlier single threaded version. As mentioned above the program now allows multiple user to operate concurrently. Thought the basic functionality of the program remains the same and user can use all file manipulation functionalities seamlessly that we introduced in our earlier design; the main changes and modification lie in the method these functionalities are now implemented in our code. The details of different function included in the design are as under:

* **Initialization Function:**

the init() function encloses 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 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.

* **Low Level Functions:**

these functions act as helpers for the function defined in the nested function called at thread creation.

* + list\_(curr)

returns children(folder/files) of the directory ‘curr’.

* + path\_(curr)

returns the path of the directory ‘curr’

* + dir\_(curr, path)

returns the directory of the specified ‘path’

* + dealloc\_(curr, name)

deallocates memory of ‘name’ (file/folder).

* + create\_(name, curr)

creates ‘name’ (file/dir) inside directory ‘curr’

* + tree\_(curr, depth)

returns as a string all children and grandchildren info of directory ‘curr’

* **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.

* **Nested Thread Function:**

this the function that is called on creation of every new thread and is passed with the id of the thread. The thread id is used to identify the user input file and output files. All the major functions are nested inside this threaded function and these functions are named after their functionalities:

* + quit() to gracefully terminate the thread
  + chdir() to change directory
  + create1() to create files
  + create2() to create directories
  + move() to perform move operation
  + delete() to perform delete operation
  + tree() to print the working directory tree
  + path() to print the path from root
  + lis() to list the content of directory
  + read() to read data from the file in file system using File class
  + write() write data to file in our file system using File class
  + append() append data to file in our file system using File class
  + File class contains methods like in previous version of file management system to simulate proper directory and file structure.

**c) Working: (Experimentation, results and observations:)**

explain working here

**d) Limitations:**

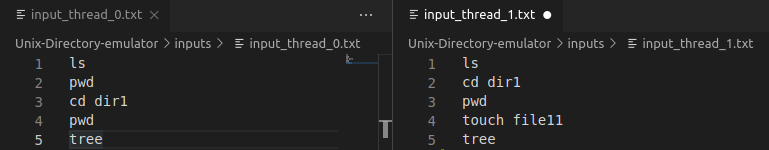
**Bellow listed are some of the limitations of our file management system.**

* **Due to the time constraints, the program is currently designed with the notion that the input files required for each user will be present in the current working directory with proper naming scheme.**
* **The file system does not support GUI and user can only check the results of their operations from the output text files.**
* **Changes to the sample.dat file can not be performed without synchronization among the user threads**
* **the metadata storage size is currently hard-coded to 1kb.**
* **In the current design there is read and write access to every user in the system and no IAM is defined.**

**e) Screenshots:**

The the screenshots are attached from a test run with two concurrent users.

Input Files:

Terminal:



Output Files:

