Group: Ernest Lin, David Tian, Justin Chan
10/6/2019
CS419
Assignment 3

Documentation

Submission Contents

Here are the names and descriptions of the files and directories included in the submission:

1. **a1_lib.py**: The library file that contains our implementation of the functions described in the assignment.
2. **a1_runner.py**: The file used to test our library.
3. **testfiles**: A directory that contains a few test files (plain text) to test our library, used by the runner program.

Testing the Library

To test our library a1_lib.py, you must run the a1_runner.py program in the following manner in the terminal:

```
    python a1_runner.py -f <path to test file> -p <path to
persistence file without file extension>
```

Example to run one of our test files, with persistence:

```
    python a1_runner.py -f ./testfiles/test1.txt -p my_data
```

Files Specified:

1. **Test file (REQUIRED)**: The actual file used to test the program
    a. Must be a plain text file (.txt extension).
    b. Each line in the test file has a command with each command component separated by a whitespace. See our submitted test files in the testfiles directory for examples.
2. **Persistence file (NOT REQUIRED)**: The file to store necessary data to allow for persistence.
    a. **IMPORTANT: When specifying the file, EXCLUDE the file extension like the example above**. Persistence is implemented using the Python pickle module, which stores the data in pickle files (.pickle extension).
    b. If not specified, the program will not save any data before it closes.

Flags:

1. f flag: Specifies the path to the test file.
2. p flag: Specifies the path to the persistence file, **but without the extension of the file**.

NOTE: All intermediate directories in the path must exist prior to running the program.

Set-Up Requirements

All you need is a test file (we provide you with the testfiles directory and three test files in it), a1_runner.py, and a1_lib.py. If you choose to store the persistence file in a subdirectory or store your own test files in another subdirectory, you must create those yourself. The persistence file itself does not need to be created by you.

Usage

The usage by a1_runner.py of each function/program in a1_lib.py is pretty straight-forward; just make a text file of commands of the same format as the examples in the assignment description (we provide 3 test file examples for you as well). As a1_runner.py executes, the results of each function/program are printed, along with the relevant data structures (users, user_groups, object_groups, access_controls).

Implementation

We store all our data in Python objects: sets, lists, tuples, and dictionaries:

- users: A **dictionary** where the key is the user name and the value is the corresponding password.
- user_groups: A **dictionary** where the key is the user group name and the value is a **set** of participating users.
- object_groups: A **dictionary** where the key is the object group name and the value is a **set** of objects in the group.
- access_controls: A **dictionary** where the key is the operation name and the value is a **set** of 2-item **tuples** that represent (user group, object group) pairs.

We used the pickle module to implement persistence because it was easy to use. Our persistence functions (store_data() and load_data()) are in a1_runner.py since there were some complications with putting them in a1_lib.py.

Additional Notes

1. We have implemented the extra credit of releasing the assumption that a user can only be in one group.
2. All user names, passwords, object names, access operations, and group names are strings without any whitespace characters.
3. When adding a user, we only check if the user name already exists and if the password is empty. If either condition is true, we throw an error.
4. When adding a user to a group, we only check if the user name already exists. If not, we throw an error.

5. When adding a user/object to a group, if the group corresponding to the input group name does not exist (and if the user exists for adding a user to a group), then the group is created and the user/object is added.
6. When adding access, we check if the user and object groups exist and return an error if either don't. If both exist and the operation specified doesn't exist, then we add the operation to the current list of operations.
7. When checking if a user can access an object, we check if the user exists, the operation exists, and the object exists (must be None or in an existing object group). If any of these conditions fail, we throw an error.