MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG445 Software Development with Scripting Languages

Assignment 2: Library Application

This assignment aims to help you practice multithreading and concurrent programming, network programming, and graphical user interface components in Python. On the successful completion of this assignment, you will also practice Python basics. Your main task in this assignment is to develop a client-server application for a library based on Transmission Control Protocol (TCP).

The client application will be used by both librarian and managers, and multiple librarian panels and manager panels should be able to be opened at the same time, so the server should be able to communicate with multiple clients at the same time as illustrated in Figure 1.

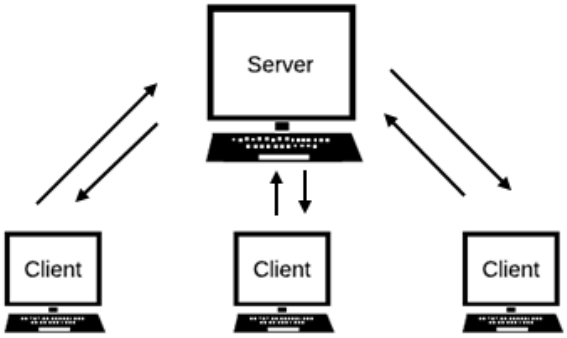

Figure 1: Client-Server Model

Overview:

In this assignment, you will develop a client-server application for a library that rents specific books. The client application will be used by the librarians and managers of the library. The librarians will be responsible for creating the operations, such as renting and returning of the books, and sending them to the server, and the managers will be responsible for requesting some statistics related to the operations, which will be generated on the server.

Books available for renting in the library are:

1. A Tale of Two Cities
2. The Little Prince
3. Harry Potter
4. And Then There Were None
5. Dream of the Red Chamber
6. The Hobbit
7. She: A History of Adventure
8. Vardi Wala Gunda
9. The Da Vinci Code
10. The Alchemist

Users can rent multiple books at the same time, for example, one person can rent Harry Potter together with The Hobbit. The same applies to returning the books as well, users can both return one rented book at a time or all of them together. By using the client application, the managers can request the following statistics related to the operations.

Both librarians and managers should login to the client application to perform their operations.

Table 1. Statistics Table

Code	Statistics
1	What is the most rented book overall? If there are multiple of them, all of them should be reported.
2	Which librarian has the highest number of operations (both renting and returning operations should be counted)? If there are multiple of them, every one of them should be reported. Please note that an operation can be a rent operation, or a return operation and it may include multiple books.
3	What is the total generated revenue by the library?
4	What is the average rental period for the "Harry Potter" book? If the book has not been rented, then 0 (zero) should be displayed. If the **recent** rental has not been returned yet, then that rental should not be counted, and the average rental period should be counted based on the other rental operations.

The server should manage three text files: **users.txt**, **items.txt** and **operations.txt.**

1. The users.txt file will keep track of the details of the librarians and managers as follows (username;password;role):

```
greg;b123;librarian
dave;k343;librarian
simon;7684;manager
```

2. The books.txt file will keep track of the books and their details as follows (bookID;title;authorName;pricePerDay;copiesAvailable):

```
1;A Tale of Two Cities;C.Dickens;2;3
2;The Little Prince;A.Exupery;2.5;2
3;Harry Potter;J.K.Rowling;2;5
4;And Then The Were None;A.Christie;2;2
5;Dream of the Red Chamber;C.Xueqin;1.5;1
6; The Hobbit;J.Tolkien;3;4
7; She: A History of Adventure;H.Haggard;2.5;2
8; Vardi Wala Gunda;V.Sharma;2;2
9; The Da Vinci Code;D.Brown;3;3
10;The Alchemist;P.Coelho;1;2
```

3. The operations.txt file will keep track of rent and return operations as follows (rent;librarianName;clientsUsername;date;itemID;itemID… or return;librarianName;clientsUsername;date;cost;itemID;itemID…):

```
rent;greg;john;05.11.2023;3;5
return;greg;john;12.11.2023;10.5;5
return;dave;john;16.11.2023;22;3
```

As you can see from the example above, client John rented 2 books with IDs 3 and 5 from the librarian Greg. However, afterwards he only returned 1 book at a time and the cost for each of the books were calculated separately, based on the date of return. Please note that this file will be populated with the operations performed by a librarian.

Requirements

- **Authentication:** When the client application is started and the connection is established with the server, the server will send a confirmation message to the client (message: **`connectionsuccess`**). Once this message is received, the client will then show the following login screen to take the username and password from the user (See Figure 2).

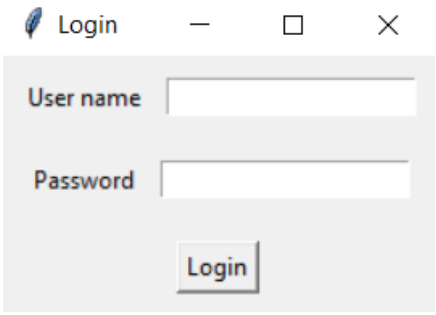

Figure 2: Login Window

Once the client sends the username and password to the server (message format: **`login;`***`username;password`*), the server will check the **users.txt** file and send an approval message (message format: **`loginsuccess;`***`username;role`*) or rejection message (message format: **`loginfailure`**) back to the client. If the login is not successful, then the error message box will be shown on the client side. However, if the login is successful, then the appropriate user panel will be shown based on the user role.

- **Librarian Panel:** Figure 3 shows the user panel for a librarian. When a librarian wants to add an operation, s/he needs to select the books, enter the date of the operation and the username of the client (Each clientsUsername is assumed to be unique, so there is no need to validate uniqueness of the nicknames explicitly). When the "**Rent**" button is clicked, the details of the operation will be sent to the server in the following format:
(**`rent;`**`librarianName;clientsUsername;date;itemID;itemID…)`.

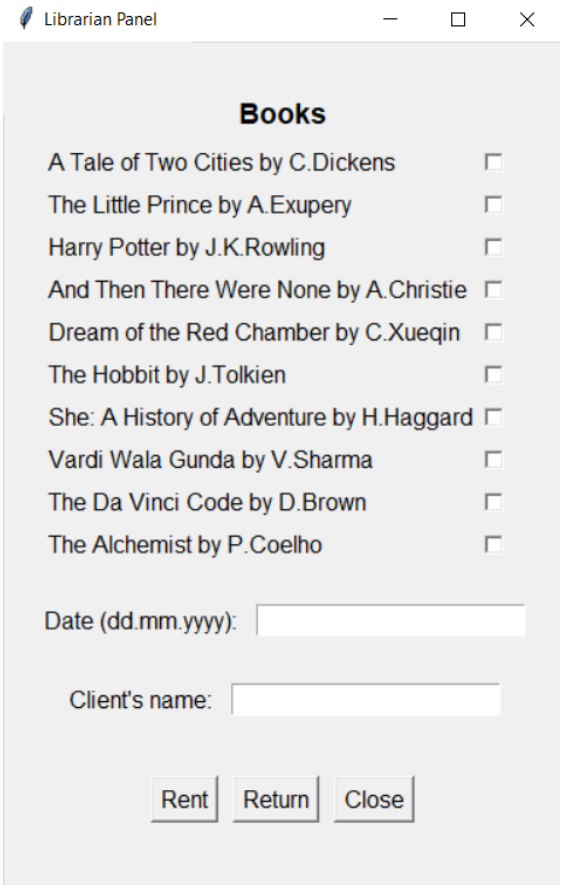

Figure 3: Librarian Panel

Upon rent operation, the server should first check the availability of the book(s). In case, the client

tries to rent multiple books at the same time, but one or more of them are not available (i.e., copiesAvailable in books.txt is 0), then error message (**availabilityerror**) should be sent back to the client application and the error message should be displayed in the message box, containing the titles of books along with their authors. Therefore, the operation will be cancelled, and no operation will be added to the **operations.txt** file.

Another validation, that must be completed prior to rent operation, is that the server should make sure that the user has already returned all the books that s/he has rented previously. If the user has not returned all the books, then the error message (**renterror**) should be sent back to client and the appropriate error message should be displayed once again, stating the names of the books, that have to be returned first. Therefore, the operation will be cancelled once again, and no operation will be added to the **operations.txt** file.

If all the above-mentioned criteria are held, then copiesAvailable field for each of the rented books should be updated in the **books.txt** file, and an operation record should be added to the **operations.txt** file and the confirmation message (**rentsuccess**) should be sent back to the client application and the success message should be displayed on the client side.

When the "**Return**" button is pressed, the details of the operation will be sent to the client in the following format: (**return**;librarianName;clientsUsername;date;itemID;itemID…). Once the return operation message is sent to the server, the server should check if the following book has been already returned or if there is no need to return the book, since it has not been rented at all by the client. In this case the error message (**returnerror**) should be sent to the client side and the appropriate error message should be displayed in the message box. If the return operation can be held successfully, then and an operation record should be added to the **operations.txt** file and the rent fee should be calculated by multiplying the daily fee to the number of days, the particular book or books were rented for, and then confirmation message should be sent to client side in the following format (**returnsuccess**;cost) and displayed once again on the client side.

For both successful rent and return operations, operations should be stored in the operations.txt, for statistics generation and rent/return validation.

When the "Close" button is clicked, the connection between the client and the server will be terminated, and the window will be destroyed.

- **Manager Panel:** Figure 4 shows the user panel for a manager. When the manager selects one of the available statistics reports, the report code will be sent to the server such as **`report2`**. The server will then generate and send back (message format: **`report2;`***`answer`* *or* **`report2;`***`answer1;answer2;…`* if there are multiple answers). The answer should be shown in a message box appropriately on the manager's side.

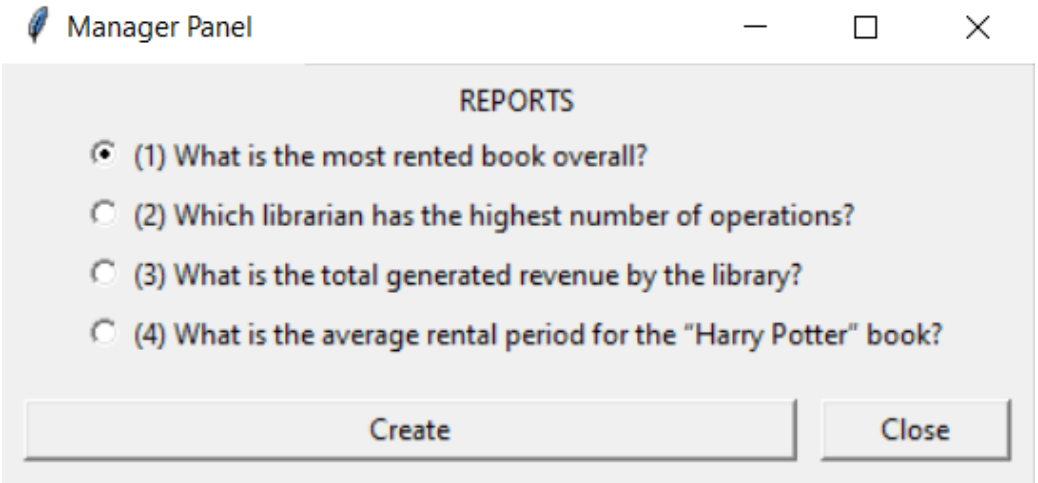

Figure 4: Manager Panel

When the "Close" button is clicked, the connection between the client and the server will be terminated, and the window will be destroyed.

Thread Synchronization: You should consider the RLock thread synchronization technique to deal with any problems caused by attempting to access the shared data at the same file.

Rules

- You need to write your program by using **Python 3.x**.
- You can **only** use all built-in functions and modules.
- You also need to create a file called ReadMe.txt which contains the following items. Please note that **if you do not submit ReadMe.txt, your submission will not be evaluated.**
    - Team members
    - Which version of Python 3.x you have used
    - Which operating system you have used
    - How you have worked as a team, especially how you have divided the tasks among the team members (who was responsible for what?), how you have communicated, how you have tested the program, etc.
- You should name your server as **server.py** and name your client as **client.py**.
- You should not forget to submit your **users.txt**, **prices.txt**, and **operations.txt** files.
- You need to put all your files into a folder that is named with your student id(s) and submit the compressed version of the folder in the **.zip** format.
- **Only one team member** should submit the assignment.
- **Code quality, modularity, efficiency, maintainability, and appropriate comments** will be part of the grading.

Grading Policy

The assignment will be graded as follows:

Grading Item	**Mark (out of 100)**
Server Side: Login	5
Server Side: Librarian Panel - Rent	25
Server Side: Librarian Panel - Return	15
Server Side: Manager Panel - Report 1	5
Server Side: Manager Panel - Report 2	5
Server Side: Manager Panel - Report 3	5
Server Side: Manager Panel - Report 4	8
Server Side: Thread Synchronization	5
Client Side: Login Window and transfer to an appropriate panel	5
Client Side: Librarian Panel	15
Client Side: Manager Panel	7