Software Design Laboratory Lab Experiments

Expt4: Event-Driven and Concurrent Programming

Content:

1. **Cover Page**

2. **PreLab**

A. **Readings, Insights, and Reflection**
METIS ebooks:
Threads:
- Core Python Programming, Rao (9789351198918) : Chapter 21
- Fundamentals of Python: First Programs, Lambert (9781337671019): Chapter 10

Graphical User Interface
- Core Python Programming, Rao (9789351198918) : Chapter 22
- Fundamentals of Python: First Programs, Lambert (9781337671019): Chapter 8

Websites:
- https://doc.qt.io/qtforpython/
- https://docs.python.org/3/library/threading.html

B. **Answer to Questions**
- Short Answer
  1. Explain what happens when a user clicks a command button in a fully functioning GUI program.
  2. Why is it a good idea to write and test the code for laying out a window's components before you add the methods that perform computations in response to events?
- Multiple Choice
  Lambert, Chapter 10, Review Questions 1 to 10

3. **InLab**
Note: Leaders should assign task to members

A. Objectives (You can state your own objectives based on the readings in Prelab)
B. Steps Performed with screenshots of tools used (VSCode/PyCharm/Spyder, Git, and QtDesigner), debugging, sample run with **DISCUSSIONS** (DON'T copy and paste from the METIS ebook)**.** Use the source codes of Lambert (BB Course Materials) and Rao.
IMPORTANT: Include figure numbers and labels. Edit your screengrabs

4. **PostLab**
Note: Leaders should assign the problems to members

A. Machine Problems

1. Write a GUI-based program that allows the user to convert temperature values between degrees Fahrenheit and degrees Celsius. The interface should have labeled entry fields for these two values. These components should be arranged in a grid where the labels occupy the first row and the

corresponding fields occupy the second row. At start-up, the Fahrenheit field should contain 32.0, and the Celsius field should contain 0.0. The third row in the window contains two command buttons, labeled >>>> and <<<<. When the user presses the first button, the program should use the data in the Fahrenheit field to compute the Celsius value, which should then be output to the Celsius field. The second button should perform the inverse function.

2. Modify the doctor application discussed in this chapter so that it tracks clients by name and history. A Doctor object has its own history list of a patient's inputs for generating replies that refer to earlier conversations, as discussed in Chapter 5. A Doctor object is now associated with a patient's name. The client application takes this name as input and sends it to the client handler when the patient connects. The client handler checks for a pickled file with the patient's name as its filename (".dat"). If that file exists, it will contain the patient's his- tory, and the client handler loads the file to create the Doctor object. Otherwise, the patient is visiting the doctor for the first time, so the client handler creates a brand-new Doctor object. When the client disconnects, the client handler pickles the Doctor object in a file with the patient's name.

3. A crude multi-client chat room allows two or more users to converse by sending and receiving messages. On the client side, a user connects to the chat room as in the ATM application, by clicking a Connect button. At that point, a transcript of the conversation thus far appears in a text area. At any time, the user can send a message to the chat room by entering it as input and clicking a Send button. When the user sends a message, the chat room returns another transcript of the entire conversation to display in the text area. The user disconnects by clicking the Disconnect button.
   On the server side, there are five resources: a server, a client handler, a transcript, a thread-safe transcript, and a shared cell. Their roles are much the same as they are in the ATM application of Project 8. The server creates a thread-safe transcript at start-up, listens for client connections, and passes a client's socket and the thread-safe transcript to a client handler when a client connects. The client handler receives the client's name from the client socket, adds this name and the connection time to the thread-safe transcript, sends the thread-safe transcript's string to the client, and waits for a reply. When the client's reply comes in, the client handler adds the client's name and time to it, adds the result to the thread-safe transcript, and sends the thread-safe transcript's string back to the client. When the client disconnects, her name and a message to that effect are added to the thread-safe transcript.
   The SharedCell class includes the usual read and write methods for a readers and writers protocol, and the SharedTranscript and Transcript classes include an add method and an __str__ method. The add method adds a string to a list of strings, while __str__ returns the join of this list, separated by newlines.

B. Debugging and Sample Run (with edited screengrabs and discussion)
IMPORTANT: Include figure numbers and labels. Edit your screengrabs

Note:

1. Save all files (.py and .ui) in one folder. Upload the folder to your OneDrive and put a tiny/bitly url in your OneNote
   Name the folder: <Course and Section>_Group#_Exp#.
   Example: **CPE106L-B1_Group1_Expt2**
2. Commit all Python source codes, QtDesigner ui files, and data files to Github (individual Github account) Github Repository Name: Software Design Lab Exercises
   Put a tiny/bitly url of your Github repository