Lab4 (Oct 13)

The objective of this lab is to practice writing multithreading programs using python.

Academic honesty and standard:

This is an individual assignment. The work you submit must be your own work. Do not share any part of your code or your design thinking. Remind yourself of the UBC policies on Academic honesty and standard.

Submissions:

Follow the requested specification exactly. We may use automated tests as a part of grading, so every unit (e.g. functions or methods) must be testable fully on own. You are allowed to use inner functions, but for consistency, do not add additional functions or methods.

You will submit a python3 file (.py extension). Submit the file to the associated Canvas assignment dropbox by the deadline.

Do not forget to include your name and student number as comments at the beginning of each .py file.

Due: **Thu Oct 13, by 7:59 PM** (common to everyone); Please plan ahead, no late submissions are accepted.

For marking, we will test for functionality first (must work), and we will check the code design, readability, compliance with the specification, documentation, .. Write the best code you can.

Lab room usage rule for Lab 4:

A TA will be in the lab during your scheduled lab time to help you, and I do recommend that you use the lab time wisely, however, for this lab we will not record attendance and there is no scheduled demo for it.

Multithreaded sorting program

In this lab we will implement a multithreaded sorting program that sorts a list of integers (assume with a length always divisible by two).

As threads within a process shared the data section, we are going to use four shared variables in our program: `testcase`, `sortedFirstHalf`, `sortedSecondHalf`, and `SortedFullList`.

The program uses three threads to complete the sorting task: two sorting threads and one merging thread. The two sorting threads use the method `sortingWorker` function. The merging thread uses

the method `mergingWorker`.

Depending on the function argument `firstHalf`, the sorting threads sort either the first half of the list or the second half of the list, and to store the result in either `sortedFirstHalf`, or `sortedSecondHalf` respectively.

The sorting is ascending, so [12, -1, 7, 7, 3, 50, 6, 8] after sorting will be [-1, 3, 6, 7, 7, 8, 12, 50]. You can implement any sorting algorithm of your choice but YOU are to code the algorithm. That is, using the *sort()* method for lists, or *sorted(),* or the like is not allowed. Document your implementation: indicate what sorting algorithm you are using and explain your code well.

Consider the following code template:

```
#student name:
#student number:

import threading

def sortingWorker(firstHalf: bool) -> None:
    """
       If param firstHalf is True, the method
       takes the first half of the shared list testcase,
       and stores the sorted version of it in the shared
       variable sortedFirstHalf.
       Otherwise, it takes the second half of the shared list
       testcase, and stores the sorted version of it in
       the shared variable sortedSecondHalf.
       The sorting is ascending and you can choose any
       sorting algorithm of your choice and code it.
    """
    pass #to Implement

def mergingWorker() -> None:
    """ This function uses the two shared variables
        sortedFirstHalf and sortedSecondHalf, and merges/sorts
        them into a single sorted list that is stored in
        the shared variable sortedFullList.
    """
    pass #to Implement

if __name__ == "__main__":
    #shared variables
    testcase = [8,5,7,7,4,1,3,2]
    sortedFirstHalf: list = []
    sortedSecondHalf: list = []
    SortedFullList: list = []

    #to implement the rest of the code below, as specified


    #as a simple test, printing the final sorted list
    print("The final sorted list is ", SortedFullList)
```

Implement the two functions first, and then complete the code under the `if __name__ == "__main__":` section.

Notes and Hints:

- use `import threading`, `threading.Thread()`, `start()` and `join()`.

- The shared variables are great help in allowing threads to work on the data and share the result. Note that the program is designed so that no two threads should work on the same shared data, so synchronization is not an issue here. The topic of synchronization is discussed extensively separately (so no need for *locks* ... in this program).
- We are not using any thread pools.
- We are using procedural paradigm for the code in this lab (to focus on threading) but it would be easy to rewrite the code using OOP if we wanted to.
- All threads are non-daemonic.
- You may want to review the "scope of variables" topic in python. There are some good examples on this in the posted Jupyter notebook that I demoed in the first lecture.
- What we are implementing here itself is a stage of a simplified/modified merge sort. Our implementation only considers the list's two halves, then each half is sorted by one thread and a final third thread sort-merges the two sorted half lists into a final sorted list.
- The original *testcase* list must not be mutated (it goes without saying).

Write the best code you can. For the marking, we check that the program works, has correct logic, follows the specs, and passes all our tests. The program should be readable (good choice of identifiers, acceptable structure and styling, useful comments wherever needed ...), and does not do any repetitive work or extra work unnecessarily.

Note: we are ignoring the fact that, as they are defined now, the threads are CPU-bound. I have kept it as such for simplicity and focus on threading. However, if, for example, the list was being provided by a networked database (like via the Internet), then it would be IO.

See the submission dropbox for the marking detail.