Project 6: Software Synchronization Solutions

Due date: Midnight of Wednesday Apr 6, 2022

Project objective:

- Understand and implement software synchronization solutions.
- Assess synchronization attempts to verify if they guarantee mutual exclusion, progress, and bounded wait-time.

Project overview:

In this assignment, you will write your own versions of:

- First synchronization “solution”
- Second synchronization “solution”
- Peterson’s synchronization solution
- Bakery’s synchronization solution

You will highlight the problems/limitations of each synchronization technique programmatically by creating well designed simulations to test mutual exclusion, progress, and bounded wait-time. Finally, you will write a **report** to showcase your work with text and figures.

The *project_6* notebook:

In the *project_6* file you will start by implementing a simple version of the race condition so you could test your synchronization solutions. In this file you are expected to create well designed simulations to test mutual exclusion, progress, bounded wait-time, and N-threads when possible. Here are the expected tests for each solution:

Technique	Mutual exclusion	Progress	Bounded wait-time	N-threads (3 or more)
First solution	✓	❌		
Second solution	✓	✓	❌	
Peterson’s solution	✓	✓	✓	
Bakery’s solution	✓	✓	✓	✓

To help you with the outline of this file, here's some code that should result in a race condition:

```python
import threading

x = 0

def increment():
    global x
    x += 1

def thread1_task(lock, my_num):
    global turn

    for _ in range(10000):
        increment()

def thread2_task(lock, my_num):
    global turn

    for _ in range(10000):
        increment()

def main_task():
    global x

    x = 0

    # create a lock
    lock = SolutionOne()

    t1 = threading.Thread(target=thread1_task, args=(lock, 1, ))
    t2 = threading.Thread(target=thread2_task, args=(lock, 2, ))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

for i in range(10):
    main_task()
    print("Iteration {0}: x = {1}".format(i,x))
```

You might need to change the number of iterations for each thread to trigger the race condition. If you don't see a race condition, choose a bigger number. Make sure the number isn't too big that it takes a long time to finish each iteration. Also, lock is created as a SolutionOne object, make sure to update that and add the code to lock/unlock the critical section in each thread. I suggest removing the lock to trigger the race condition before you do anything.

The *solution_one.py* file:

In the *solution_one.py* file you will implement the first synchronization attempt from lecture slides (lecture 12) using a Python **class** that implements the following:

- An *__init__* method that initializes a *turn* variable.
- A lock method that takes a *thread_id* as an argument. The method should behave according to the pseudocode in lecture slides.
- An unlock method that takes a *thread_id*, and uses it to change the value of turn according to the pseudocode in lecture slides.

After implementing this simple synchronization attempt, import it and test mutual exclusion and progress in your project_6 notebook. Be creative in testing progress, and I am happy to verify your ideas via email or during office hours if you like.

The *solution_two.py* file:

In the *solution_two.py* file you will implement the second synchronization attempt from lecture slides (lecture 13) using a Python **class** that implements the following:

- An *__init__* method that initializes a *flags* list.
- A *lock* method that takes a *thread_id* as an argument. The method should behave according to the pseudocode in lecture slides.
- An *unlock* method that takes a *thread_id*, and uses it to change the value of *flags* according to the pseudocode in lecture slides.

After implementing this synchronization attempt, import it and test mutual exclusion, progress, and bounded wait-time in your project_6 notebook. Be creative in testing bounded wait-time, and I am happy to verify your ideas via email or during office hours if you like. As a hint, you can use the following code to trigger a context switch (you can add the line anywhere in your code in your bounded wait-time simulation): **time.sleep(0.0001)**

The *petersons_solution.py* file:

In the *petersons_solution.py* file you will implement Peterson's synchronization solution from lecture slides (lecture 13) using a Python **class** that implements the following:

- An *__init__* method that initializes a *flags* list and a *turn* variable.
- A *lock* method that takes a *thread_id* as an argument. The method should behave according to the pseudocode in lecture slides.
- An *unlock* method that takes a *thread_id*, and uses it to change the value of *flags* according to the pseudocode in lecture slides.

After implementing this synchronization attempt, import it and test mutual exclusion, progress, and bounded wait-time in your project_6 notebook.

The *bakery_solution.py* file:

In the *bakery_solution.py* file you will implement Bakery's synchronization solution from lecture slides (lecture 14) using a Python **class** that implements the following:

- An *__init__* method that takes a thread_count and initializes a *choosing* list and a *tickets* list.
- A *lock* method that takes a *thread_id* as an argument. The method should behave according to the pseudocode in lecture slides.
- An *unlock* method that takes a *thread_id* and behaves according to the pseudocode in lecture slides.

After implementing this synchronization attempt, import it and test mutual exclusion, progress, bounded wait-time, and **n-threads** in your project_6 notebook.

Report:

Organize your notebook report as follows, maintaining a coherent document that includes screenshots and text to communicate the objective of your project:

1. Abstract: A brief summary of the project (including findings), in your own words. This should be no more than 150 words. Give the reader context and summarize the results of your assignment.

2. Results: A section that goes over the code you implemented and the performance of each synchronization solution.

3. Discussion: A section that interprets and describes the significance of your findings focussing on the results.

4. Extensions: Describe any extensions you undertook, including text output, graphs, tables, or images demonstrating those extensions.

5. References/Acknowledgements.

Extensions:

You can be creative here and come up with your own extensions. I will suggest the following ideas:

- Implement your own synchronization solution and test it.
- Modify Peterson's solution to work with n-threads (it is possible) and test it.

Project submission:

- Add all your files (except *.ipynb_checkpoints* and *__pycache__*) to your ***project_6*** directory on Google Drive.
- Copy the rubric from Moodle to the project directory after you review it.