Queen's School of Computing
CISC324 – Fall 2024
Instructor: Dr. Anwar Hossain
Lab 3 - Due Date: March 4, 2024

Lab 3

This lab has been prepared to be conducted on any operating system (Linux, MacOS, and Windows). Please make sure to have a well-set-up environment. You will need a computer, any operating system, and any Python interpreter of your choice. Note, python is chosen intentionally.

Background and Outcomes

This lab assignment explores the classic synchronization problem known as the Producer-Consumer problem. You will implement a solution using Python's threading and semaphore mechanisms to manage access to a shared buffer. Producers will add data to the buffer, and consumers will remove data, with the challenge being to ensure that producers do not add to a full buffer and consumers do not remove from an empty one.

Objective

Implement a threaded program in Python that solves the Producer-Consumer problem using semaphores for synchronization. You will fill in missing parts of a provided code skeleton to ensure that producers and consumers operate correctly without interference and avoid conditions such as deadlock or data inconsistency.

Boilerplate Code

We've provided you with a starter code that includes the basic structure of the producer-consumer problem.

Your tasks for this lab include completing the '**TODO**' sections in the provided code skeleton. Specifically, you will:

- Implement waiting logic for producers when the buffer is full.
- Signal consumers when data is added to the buffer.
- Implement waiting logic for consumers when the buffer is empty.
- Ensure that consumers properly handle the scenario when production is done and the buffer is empty.
- Signal producers when data is removed from the buffer, indicating space availability.
- Correctly implement the end-of-production signaling to allow for graceful termination of consumer threads.

Queen's School of Computing
CISC324 – Fall 2024
Instructor: Dr. Anwar Hossain
Lab 3 - Due Date: March 4, 2024

Detailed Code Descriptions

SharedBuffer Class

This class represents the shared buffer between producers and consumers. It includes methods for safely adding and removing messages, with synchronization mechanisms like locks and semaphores to ensure correct producer and consumer operations without data corruption or loss.

add_message(self, message)

Allows a producer to add a message to the buffer. If the buffer is full, it waits until there's space. Signals that the buffer is not empty afterwards.

read_message(self)

Allows a consumer to read and remove a message from the buffer. If the buffer is empty, it waits until there's a message. Signals that the buffer is not full afterwards.

mark_done_producing(self)

Sets a flag indicating that producers have finished adding messages and notifies consumers potentially waiting on an empty buffer.

check_done_producing(self)

Checks if the production has finished and the buffer is empty, indicating that consumers should stop waiting for new messages.

Producer Function

Simulates a producer that generates and adds messages to the shared buffer. It ensures that the buffer is not overwhelmed by waiting if necessary and notifies consumers upon adding new messages.

Consumer Function

Simulates a consumer that retrieves and processes messages from the shared buffer. It waits for messages to be available and respects signals indicating production completion.

Main Function

Coordinates the initialization and execution of producer and consumer threads based on predefined simulation parameters. It ensures that all threads start and complete their execution properly.

Simulation Parameters

The simulation involves adjustable parameters such as buffer size, number of producers, number of consumers, read time, and write time. Put in your code an option to vary these parameters. These parameters allow you to test the robustness of your solution under various conditions.

Bonus question: Can you extend your solution to handle priority-based consumption, where certain consumers have higher priority over others in consuming the produced items.

Queen's School of Computing
CISC324 – Fall 2024
Instructor: Dr. Anwar Hossain
Lab 3 - Due Date: March 4, 2024

Describe how you would modify your synchronization mechanism to accommodate this requirement.

Prepare your environment!

This code can work on any environment (Linux, Windows, MacOS, etc.) so please feel free to search for how to install Python on your machine.

IDEs that you can use (feel free to use any other IDE)

1. Visual Studio Community Edition for Windows-based machines only (it's something different from VS Code).
2. Visual Studio Code (Can work on any operating system).
3. PyCharm (Can work on any operating system).

Install Python on Your Machine

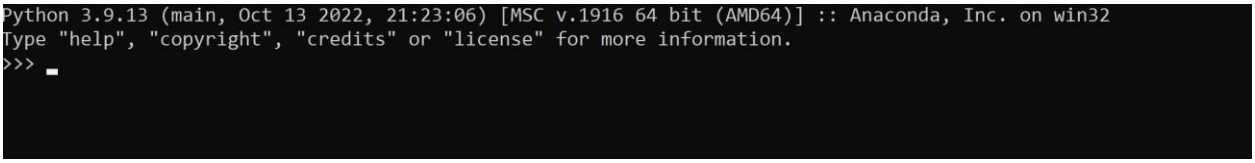
There are too many methods to install Python (feel free to use any other method), ignore if done earlier.

1. Download and install Anaconda (which works on any operating system).
2. Download and install Python from the official website (works on any operating system).

Required Libraries

We will use "threading" library to implement threads and use semaphores. After installing Python using either method mentioned, make sure that Python is working properly on your machine.

1. Open your command prompt or terminal if you're using Mac/Linux.
2. Type the command python.
3. Check the output on the console screen. You should be seeing something like that:



   If you don't see the python version or if you encounter any errors, then there might be something wrong with the installation.
4. Install "threading" library by executing this command in your command prompt/terminal:
   pip install thread6

Queen's School of Computing
CISC324 – Fall 2024
Instructor: Dr. Anwar Hossain
Lab 3 - Due Date: March 4, 2024

Download the Boilerplate Code

There are two methods to download the boilerplate code:

1. Download the whole repository of the course (Zip file) and open the folder named "Lab 3".
2. If you are familiar with Git and Git commands, you can clone the repository or you can update your local repository if you already have it cloned from the previous lab.

What to submit?

1. Place all your source codes and the readme.txt file in a folder named in the following format:

   324-group**X**-Lab**3**, where X stands for the group number, e.g.: example folder name for group 1 is 324-group**1**-Lab**3**.

   Note: The **readme.pdf** should report the results of the execution.

2. Compress the above folder using Zip (the extension must be .zip or .rar).
3. Log into OnQ, locate the lab's dropbox in OnQ, and upload the compressed folder