PYTHON

Assignment

Task 1: Basic Multithreading

- Create a Python program that launches two threads. One thread prints numbers from 1 to 5, while the other prints numbers from 6 to 10.
- Ensure both threads run concurrently.

Task 2: **Thread with Arguments**

- Write a Python program where a thread accepts a list of numbers as an argument and calculates the sum of the numbers.
- Create a second thread that accepts another list of numbers and calculates the product.

Task 3: **Daemon Thread**

- Write a Python program where a non-daemon thread prints "Main thread finished," and a daemon thread continuously prints "Daemon running..." every second. Ensure that the daemon thread terminates automatically when the main thread finishes.

Task 4: **Thread Sleep**

- Create a Python program where a thread prints the current time every 2 seconds for 10 iterations.
- Introduce a second thread that prints "Still running..." every 5 seconds.

Task 5: **Thread Synchronization with Locks**

- Write a Python program where two threads modify a shared global counter. Ensure that both threads increment the counter 100 times.
- Use a `Lock` to synchronize access to the counter and prevent race conditions.

Task 6: **Thread Synchronization with** `RLock`

- Write a Python program using a recursive lock (`RLock`). Create a function that uses multiple `RLock` acquisitions inside a thread and prints messages at each level of the recursion.

Task 7: **Producer-Consumer Problem**

- Implement a basic producer-consumer problem using threads and a shared queue.
  - The producer should add random numbers to the queue.
  - The consumer should take numbers from the queue and print them.
- Use thread synchronization to ensure proper access to the queue.

Task 8: **Thread Synchronization with** `Semaphore`

- Implement a Python program where a group of threads attempts to access a limited resource, such as a connection pool with 3 connections.
- Use a `Semaphore` to ensure that no more than 3 threads can access the resource simultaneously.

Task 9: **Thread Synchronization with** `Event`

- Create a Python program using an `Event` object.
  - One thread should wait for a signal from another thread using the `Event` mechanism.
  - The first thread should only start its task once the second thread sets the event after completing some initial work.

Task 10: **Thread Termination and Exception Handling**

- Create a Python program where a thread performs a long-running task, such as calculating prime numbers indefinitely.
- Implement a mechanism to gracefully terminate the thread after a set amount of time (e.g., 10 seconds), and handle any exceptions that may occur during the thread's execution.