Faculty of Science, Engineering and Technology

Concurrent Programming

Pass Task 04: Semaphore

Overview

Threads need means of communicating. The simplest utility to support this is the Semaphore, otherwise known as a counting semaphore.

Pass Task 04 — Submission Details and Assessment Criteria

You must submit the following files to Doubtfire:

- Semaphore source code (Semaphore.cs)
- Semaphore Test class (Test.cs)

Instructions

Create a Semaphore class in your *concurrent utilities library*. Implement the following features:

- A protected integer (or long) **_count** field.
- Constructor(n) initialise the Semaphore with n tokens. Default n to 0.
- **Release** calls Release(1)
- **Release(n)** adds *n* to the _count and pulses all threads waiting for a token.
- **Acquire** waits for a token to be available, then reduces the _count by 1 and returns.
- Ensure that all public features, including the class, have XML documentation.

Create a Semaphore Test class that demonstrates the use and features of a Semaphore. In this case the features you want to demonstrate are:

- Threads cannot acquire until a token arrives.
- When multiple threads are waiting to acquire, and one token is released, only one of the waiting threads proceeds.
- When multiple threads are waiting to acquire, and multiple tokens are released, the number of threads that proceed equal the number of tokens released.
- When a token is released to a Semaphore with no waiting threads, the next thread to Acquire will proceed without waiting.

SWINBURNE UNIVERSITY OF TECHNOLOGY

Note: Make sure the output show if the Semaphore is/isn't working correctly. Lots of fast output is bad!