Practica 3

Paweł Jędrzejczyk

Wojciech Pacześniak

Eje 1

Time when using semaphores: 5.886

Time when using Queues: 6.938

Using Queues introduces some overhead that can impact performance compared to the direct manipulation of lists and semaphores used in the version with Semaphores.

In this exercise we used queue.Queue instead of semaphores. Thanks to that code is simpler and easier to read.

Eje 2

Time when using Lock: 6.370

In the version without Lock, all threads share a single queue (queue.Queue) for communication between the producer and consumer threads. This can lead to contention as multiple threads compete for access to the queue, resulting in locking and potential delays. In the second version, the use of a threading lock (lock) ensures that only one thread accesses the shared variables at a time, reducing contention and improving performance.

In second exercise we added lock in order to improve synchronisation. The use of a threading lock enhances the reliability and correctness of the multi-threaded application by ensuring that shared variables are accessed and updated safely by multiple threads.

Eje 3

In this exercise we used Barrier and Semaphores in order to count molecules so when water molecule are being created correctly. Barriers provide a

synchronization point where threads must wait until a specified number of threads have reached the barrier. In this case, the barrier is set to 3, indicating that each molecule of water requires two hydrogen atoms and one oxygen atom. Once all three atoms have arrived at the barrier, the barrier action (moleculagenerada) is executed, indicating that a molecule of water has been successfully created.

Eje 4

Events provide a simple way to synchronize the activities of multiple threads. In this case, the reponer_event is used to signal to waiting client threads that the product has been restocked. This ensures that clients only attempt to purchase the product when it's available, preventing race conditions and inconsistencies.