CONCURRENCY
Assignement 04

16-124-836
Marcel ZAUDER

# 1 CASLock and CCASLock

## 1.1 Execution with 4 Threads

| | **CASLock** | **CCASLock** |
|---|---|---|
| **Counter Value** | 300'000 | 300'000 |
| **Executiont Time** | 39ms | 27ms |
| **Lowest Number of Accesses** | ∼ 50'000 | ∼ 35'000 |
| **Highest Number of Accesses** | ∼ 100'000 | ∼ 155'000 |

## 1.2 Execution with 8 Threads

| | **CASLock** | **CCASLock** |
|---|---|---|
| **Counter Value** | 300'000 | 300'000 |
| **Executiont Time** | 60ms | 51ms |
| **Lowest Number of Accesses** | ∼ 9'000 | ∼ 7'000 |
| **Highest Number of Accesses** | ∼ 75'000 | ∼ 85'000 |

## 1.3 Peterson's Algorithm - 8 Threads (*Comparison*)

| | |
|---|---|
| **Counter Value** | 300'000 |
| **Executiont Time** | 164ms |
| **Lowest Number of Accesses** | ∼ 32'000 |
| **Highest Number of Accesses** | ∼ 41'000 |

## 1.4 Conclusion

In all executions the counter did not exceed the value of 300'000 and all accesses to the critical section sum up to 300'000 as well, which leads to the conclusion that the three different locks work correctly. The differences between the *Peterson's Algorithm* and the newly implemetned *CASLock* and *CCASLock* lies within the fairness which was fairly given in the implementation of the former. The values between the lowest and highest access numbers in the algorithms of *CASLock* and *CCASLock* differ tremendously.
Furthermore we can see that the fairness comes with a huge overhead which leads to a slightly higher execution time for the *Peterson's Algorithm*, whereas the *CASLock* and *CCASLock* are much faster.
When comparing the 4 threads and 8 threads execution of *CASLock* and *CCASLock* we can conclude that the more threads are involved the more overhead and waiting time is created which affects the execution time to be slower for the execution with more threads.

## 2 Unbounded Lock

The necessaty for a *non-empty queue* check in the *deq()*-method follows from the following otherwise occuring problem:

- *queue* has **one** element enqueued
- *Thread B* checks for empty queue → FALSE
- *Thread B* enters critical section
- *Thread A* checks for empty queue → FALSE
- *Thread B* dequeues the last element in the *queue* inside critical section
  ⇒ *queue* is now empty
- *Thread A* enters critical section
- *Thread A* tries to dequeue from an empty queue

## 3 Queue