NWI-IBC019: Operating system concepts / Exercises: week 4 / 2022 v0

NOTE *These weekly exercises are individual (unless marked otherwise).*

READING **10TH EDITION** *section 7.1 (except for the part on monitors) and 7.5.*

HAND IN *Please hand in the exercises in PDF format in Brightspace.*

Exercise 1

Due to the pandemic, the restaurant "Breadlock" needs to make stricter rules. Since people need to keep distance, at most 5 customers are allowed to be in the restaurant simultaneously. Furthermore, after a customer leaves, the table needs to be cleaned by the cleaner. Once the table is clean, a new customer is allowed to enter.

Implement threads for the customers and the cleaner and argue that your solution does not contain deadlocks. The thread for the customers invokes the functions eat and leave, while the thread for the cleaner only invokes the function clean.

Exercise 2

During the lecture, we discussed the reader-writer problem. In this exercise, we look at a different solution to this problem. Instead of treating readers and writers the same way, we give priority to the writers. So, once a writer would like to write, all readers that aren't reading at that moment, have to wait until that writer is finished.

- Give an implementation of the described algorithm.
- Is starvation possible in your solution? If so, describe the scenario in which starvation could happen. If not, explain why it is not possible.

You can assume you have the following shared data

```
int readers = 0;
semaphore m_readers = {1};

int writers = 0;
semaphore m_writers = {1};

semaphore readersAllowed = {1};
semaphore writersAllowed = {1};
```

Exercise 3

The goal of this exercise is to compute the first n prime numbers in parallel. You can assume that each thread performs the same task T and that the following data is shared between them:

```
const int n = 10000;
int next = 0;
bool primes[n];
```

The constant n gives the upper bound, and next is the integer that will be checked next. Lastly, the array primes says which numbers are prime.

Implement the function T. You can decide yourself whether you use semaphores or mutexes. In your implementation, you should clearly list the initial values of the synchronization primitives you use. You may use the function isPrime, which is implemented as follows

```
bool isPrime (int n) {
  if (n == 0 || n == 1) {
    return false;
  }
  for (int i = 2; i < n; i ++) {
    if (n % i == 0) {
    return false;
  }
  return true;
}
```