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

READING **10TH EDITION** *chapter 8 up to and incl. 8.5.*

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

Exercise 1

Now let us go to the local bakery where there is a baker and a number of customers. We assume that we have semaphores `sem` and `items`, we have some collection `shelves`, and we have functions `bakeBread` and `buy`. The baker behaves according to the following thread:

```
1  b = bakeBread();
2  sem.wait();
3  shelves.add(b);
4  items.signal();
5  sem.signal();
```

The behavior of the customers is described by the following thread:

```
1  sem.wait();
2  items.wait();
3  b = shelves.get();
4  sem.signal();
5  buy(b);
```

- What is the point of the semaphore items?
- Show that this solution contains a deadlock and visualize the deadlock using a resource allocation graph;
- Give an improved solution and argue that your version is free of deadlocks.

Exercise 2

Let's return to dining philosophers. However, for some reason two left, and we now only look at three dining philosophers. Call the philosophers `P0`, `P1`, and `P1`. Furthermore, we have three forks, which are implemented as semaphores `f0`, `f1`, and `f2`. The three philosophers behave as follows

```
void P0() {
  while (true) {
    f0.wait();
    f2.wait();
    eat();
    f2.signal();
    f0.signal();
  }
}
```

```
void P1() {
  while (true) {
    f2.wait();
    f1.wait();
    eat();
    f1.signal();
    f2.signal();
  }
}
```

```
void P2() {
  while (true) {
    f0.wait();
    f1.wait();
    eat();
    f1.signal();
    f0.signal();
  }
}
```

- Argue whether a deadlock is possible in this system.
- Argue whether a deadlock is possible if we swap the order of some of the `wait` operations.
- Argue whether a deadlock is possible if we swap the order of some of the `signal` operations.

Note: if no deadlock is possible, you should give a reason. You don't need to give a resource allocation graph if you want to show there's a deadlock.

Exercise 3

We look at the cigarette smokers problem. In this problem, there are three smokers, who all need matches, paper, and tobacco for their cigarettes. One smoker has matches, one has paper, while the last one has tobacco. None of them is able to actually smoke, because they lack two of the three necessary ingredients. However, instead of sharing with each other, they ask an agent to provide them what they are missing. The agent does not listen carefully and instead of giving them exactly what they want, two random ingredients are chosen.

The goal of this exercise is to implement threads performing these tasks. You can assume that you have the following share data:

```
agentSem = Semaphore(1);
tobacco = Semaphore(0);
paper = Semaphore(0);
match = Semaphore(0);
```

To implement the agents, we use a trick. Instead of using a randomizer, we implement three threads: one for each possibility. When the agent is called, one is these threads is picked at random, and thus the agent satisfies the required specification.

```
void agentA() {
  while (true) {
    agentSem.wait();
    tobacco.signal();
    paper.signal();
  }
}
```

```
void agentB() {
  while (true) {
    agentSem.wait();
    paper.signal();
    match.signal();
  }
}
```

```
void agentC() {
  while (true) {
    agentSem.wait();
    tobacco.signal();
    match.signal();
  }
}
```

- Provide an implementation for the smokers.
- Does your solution have a deadlock? If there is a deadlock, describe how it arises (possibly using a resource allocation graph). If there is not a deadlock, you don't need to give a reason.