Deadlock prevention

What are the necessary conditions for deadlocks

- *Mutual exclusion*: Limited number of threads may utilize a resource concurrently.
- *Hold and wait*: A thread holding a resource may request access to other resources and wait until it gets them.
- *No preemption*: Resources are released only voluntarily by the thread holding the resource.
- *Circular wait*: There is a set of threads, where T1 is waiting for a resource held by T2. T2 is waiting for a resource held by T3 and so forth.

Why does the initial solution lead to a deadlock (by looking at the deadlock conditions)?

The initial solution leads to a deadlock because a **circular wait** happens. The problem is that the right fork of one philosoph is the left fork of the other philosoph and the left fork is taken first.

Example: 3 Philosophers For example a philosoph p1 claims the left fork f1, suddenly the execution is switched to another philosoph p2, which takes his left fork f2, which is the right fork f2 of p1. Therefore, p1 occupies the fork f1, until the p2 releases f2. p2 can't take his right fork f1 afterwards, which results in a **circular wait**. p3 can take his left fork f3, but can't also not continue, because f2 is occupied by p2. Therefore, the problem also contains **Hold and wait**, because a thread requests the left fork first and then waits for the right fork. **Mutual exclusion** is also the case because a limited number of threads (philosophers) want to a resource concurrently (forks). And **No preemption** is also fulfilled, because only the thread releases the occupied forks.

Does this strategy resolve the deadlock and why?

The deadlock is resolved because the fork between odd and even philosophers can't be blocked. An odd or even philosopher can always take the fork between two philosophers.

The odd philosopher takes his left fork first, while the even philosopher takes his right fork. The fork between the philosophers can be taken by both philosophers, because it is the right fork of the odd philosopher and the left fork of the even philosopher.

Therefore, at least one philosopher can take both forks.

Measure the total time spent in waiting for forks and compare it to the total runtime. Interpret the measurement - Was the result expected?

I tested the runtime in a few runs. With philosophers between 5-8 and a thinking- and eating-time of 500-1000 ms. 5 of the runs are documented inside the `philosophers-test.xlsx`.

As seen in the evaluation, the time of waiting for forks is about 31% - 35%. Waiting for the right fork takes about 40% -50% of the time waiting for the left fork. These measurements were taken from the summed up runtime of every thread and the summed up time waiting for forks.

The results were pretty surprising for me, because I was not thinking that waiting for forks makes about one third of the total runtime of the program. My expectations were about 10-15%, because the improved version seemed to eliminate the problem with long waiting times