Gabriel Wiedermann, Strauß David - MSE-2A

Exercise 1, Concurrency – Dining Philosophers

Why does the deadlock occur?

What are the necessary conditions for deadlocks (discussed in the lecture) [0.5 points]?

There are 4 conditions which are needed to be fulfilled:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

Why does the initial solution lead to a deadlock (by looking at the deadlock conditions) [0.5 points]?

mutual exclusion -> Each thread locks the fork which it is using

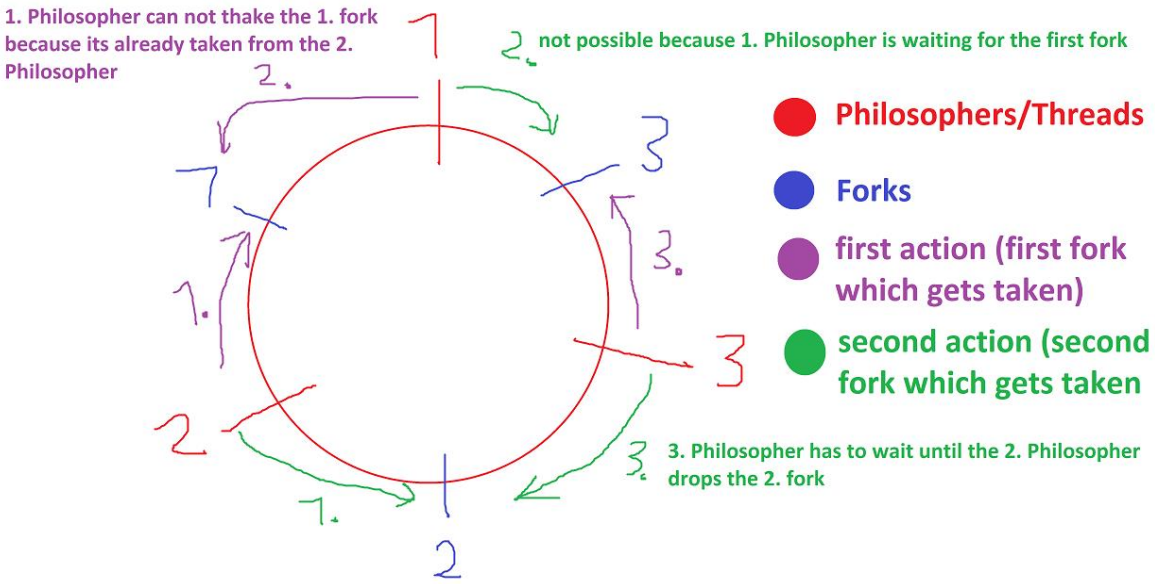
Hold and wait -> Our threads are using the resource forks (array of forks) and a thread has first to request access for the left fork and afterwards he is requesting access for the right fork. Due to this logic the thread is holding the left fork and is waiting until the right fork gets free.

No preemption -> The other philosophers can’t steal the forks and they hold it until they are finished with their task

Circular wait -> We have an amount of philosophers (Threads) and each philosopher tries to get his left fork and then he wants to get his right fork but his right fork is the left fork for another philosopher. Due to this we are facing a circular wait.

Gabriel Wiedermann, Strauß David - MSE-2A

Prevent the deadlock by removing Circular Wait condition - done (check code)
Does this strategy resolve the deadlock and why [1 point]?
Yes it does because if you can ensure that at least one of the four before mentioned condition is not fulfilled a deadlock is not possible. In this case we ensured that Circular wait can not happen because now this situation described before can not happen.Because now the odd Thread takes first the left fork but his neighbour is an even Thread and thats why he takes the right fork. Due to that they are not competing about the same fork.



Of course in this picture we have chosen a random order for the threads but it still shows/explains why it can not end up in a deadlock.

Measure the overall time, philosophers spend for eating. How does it compare to the overall time execution time of the dinner

The time philosophers spend eating is of course dependent on the input parameters. For example with input parameters [numberOfPhilosophers = 5, maxEatingTime = 5, maxThinkingTime = 5] we get this output:

Total eating time was: 10250
Eating time per philosopher was: 2050
Total dinner time was: 6683

So the eating time per philosopher makes up about one third of the runtime. If we play around with the input parameters, the ratio changes.

Can you think of other techniques for deadlock prevention?
Using TryEnter to get rid of Hold and Wait condition. If the Philosopher does not get the resources he needs during his wait time, he frees his resources for others.

```
var lockObj = new Object();
var timeout = TimeSpan.FromMilliseconds(500);
bool lockTaken = false;

try {
   Monitor.TryEnter(lockObj, timeout, ref lockTaken);
   if (lockTaken) {
      // The critical section.
   }
   else {
      // The lock was not acquired.
   }
}
finally {
   // Ensure that the lock is released.
   if (lockTaken) {
      Monitor.Exit(lockObj);
   }
}
```

One further idea would be to make preemption possible -> for example lets say one philosopher is more important than the rest he can Pulse the ressource he needs.