

## Excercise Sheet 9

### Solution

## Lecture Distributed Systems

### Winter Term 2025/26

#### Exercise 1: Ring Algorithm

When a process receives an ELECTION message, it checks who sent it. If it has sent it itself (i.e. its number is at the beginning of the list), it converts the message into a COORDINATOR message, as described in the lecture. If it has not sent an ELECTION message, it enters its process number and passes the message on in the ring.

However, if it has previously sent his own ELECTION message and has just discovered a competitor, it compares the process number of the originating process with its own. If the other process has a lower number, it discards the message instead of passing it on.

If the other process has a higher number, the message is forwarded as usual. In this way, only the ELECTION message with the higher number remains when multiple ELECTION messages are sent. The rest is discarded along the way.

#### Exercise 2: Centralized Mutual Exclusion Algorithm

- a) Requests could be assigned priority levels depending on their importance. The coordinator could then serve the highest priority request first.
- b) Suppose the algorithm specifies that each request is answered immediately, either with a permission or with a denial. If there are neither processes in a critical section nor in the queue, a crash is not fatal.

The next process requesting a permission receives no response and can then initiate the election of a new coordinator. The system can be made even more robust by allowing the coordinator to save each incoming request to disk before returning a response. In this way, in the event of a crash, the new coordinator can reconstruct both the list of active critical sections and the queue by reading the file from disk.

#### Exercise 3: Algorithm of Ricart and Agrawala

Suppose a process denies permission and then crashes. The requesting process believes it is still alive, but the permission never arrives. A way out is to cause the requester not only to block, but to *sleep* for a certain amount of time, after which he queries all processes that have denied permission to check if they are still running.

#### Exercise 4: Deadlocks - Ricart and Agrawala

That depends on the fundamental rules. If the processes enter into critical sections strictly sequentially, i.e. a process in a critical section must not attempt to enter into another critical section, there is no way it can block while holding a resource (i.e. a critical section) that another process needs. The system is deadlock-free. On the other hand, if process 0 can enter critical section A and then attempt to enter critical section B, a deadlock may occur if another process attempts to enter the critical sections in reverse order. The algorithm of Ricart and Agrawala itself does not contribute to the deadlock because each critical section is treated independently of the others.