TUM I 13
Chair of Application and Middleware Systems
Prof. Dr. H.- A. Jacobsen

Tutorial
Distributed Systems (IN2259)
WS 2020/21

SAMPLE SOLUTION: EXERCISES ON COORDINATION & AGREEMENT

Note: this sample solution is only a suggestion to solve the assignment; there might be various other possible solutions. Also note that this sample solution might still contain errors or other fallacies.

EXERCISE 1 Multicast-based Mutual Exclusion

Does the following algorithm solve the mutual exclusion problem? Discuss your answer based on the requirements for the solutions of the mutual exclusion problem, as discussed in the lecture.

```
On initialization
       state := RELEASED;

To enter the section
       state := WANTED;
       Multicast request to all processes;
       Wait until (number of replies received = (N - 1));
       state := HELD;

On receipt of a request Pi at Pj (i!=j)
       if (state = HELD or (state = WANTED and Pj < Pi))
       then
                      queue request from Pi without replying;
       else
                      reply immediately to Pi;

       end if

To exit the critical section
       state := RELEASED;
       reply to any queued requests according to the current status of process;
```

Solution:
Requirements for Mutual Exclusion:

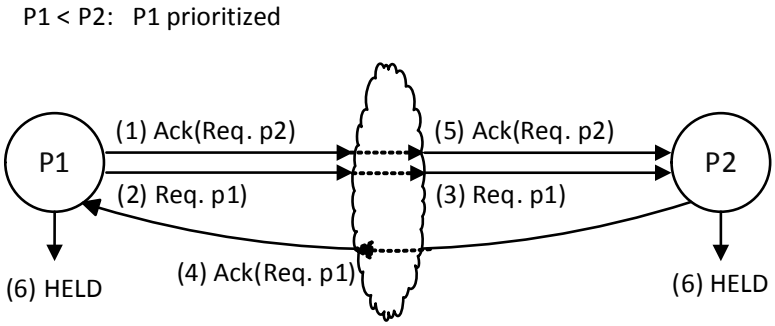
- **Safety** At most one process accesses the CS at a time
- **Liveness**
  - Requests to enter & exit the CS eventually succeed
  - No deadlocks
- **Fairness**
  - If one request to enter the CS happened-before another one, then access is granted in that order

– Requests are ordered such that no process enters the CS twice while another process waits to enter (no *starvation*)

This algorithm satisfies none of the requirements of the mutual exclusion problem.

Safety is not guaranteed. For instance, suppose there are only two processes P1 and P2 in the system:

1. P1 is in the CS and P2 is waiting for receiving a reply message from P1 to access the CS
2. P1 leaves the CS and sends a reply message to P2
3. Directly after sending the reply message, P1 again sends a new request message
4. P1 gets the reply from P2 (because P1's new request has overtaken P1's reply. This was not detected by P2 due to the lack of a notion of time)
5. Now P2 receives the reply message from P1
6. P1 changes its state to HELD and goes into the CS
7. P2 changes its state to HELD and goes into the CS



→ There are now two processes in the CS; therefore, safety is violated.

Liveness is violated because for every two processes P_j and P_i with $j < i$, if process P_j is in state WANTED or HELD, it will never allow process P_i to get in the CS. This is the case when P_j continuously requests access.

Fairness fails because if P_i requests to enter the CS before P_j but $j < i$, then P_j will enter the CS first.

Finally, the *Ordering* property is not guaranteed, since the algorithm is based only on process ids, and a notion of time is ignored. Consequently, the happened-before relation cannot be obtained.

EXERCISE 2 Ring-based Mutual Exclusion

The original algorithm requires the token to be passed continuously even when no process desires access to the critical section, thereby wasting communication resources. Propose a new algorithm where Chang & Roberts algorithm is integrated with the ring-based approach in a way that communication cost is only required when there is at least one process that requires access to the critical section. Assume an asynchronous model with reliable communication, FIFO channels, and no process failure. Discuss if your proposed solution is safe?

Solution:

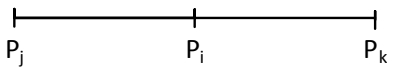
```
On initialization
       state := RELEASED;
       vectorClock_i := makeEmptyClock();
       token_i := makeToken(Pi);
       queue := null;

To enter the section at Pi
       state := WANTED;
       UpdateClock(vectorClock_i)
       msg := (vectorClock_i, token_i);
       Send msg to P(i+1) mod N;
       Wait until msg with token_i is received from P(i-1) mod N;
       state := HELD;

On receipt of a msg (vectorClock_j, token_j) at Pi (i!=j)
       if (state = HELD) or (state = WANTED and vectorClock_j > vectorClock_i) or (state =
           WANTED and vectorClock_j || vectorClock_i and token_j > token_i)
       then
                      put msg into queue;
       else
                      UpdateClock(vectorClock_i, vectorClock_j)
                      send msg (vectorClock_i, token_j) to P(i+1) mod N;
       end if

To exit the critical section at Pi
       state := RELEASED;
       update vector clocks of msgs in queue and send msgs to P(i+1) mod N;
```

Safety is guaranteed because each process has one token and vector clock and the messages are processes based on vector clock.

EXERCISE 3 Leader Election

(a) For leader election, assume that two processes detect the demise of the coordinator simultaneously and both decide to hold an election using the Bully algorithm. What happens?

Solution:



- Example: $j < i < k$ (cf. above)
- @P_i : Broadcast *election*
- P_k sends *answer* (higher id) and broadcasts new *election*, P_j doesn't answer (lower id)
- if P_i receives no *answer* it broadcasts *victory* else P_i waits for *victory*.

→ Each of the higher-numbered processes will get multiple *election* messages, but will ignore those it received after starting its own election round. The election will proceed as usual.

(b) How is the Bully algorithm dealing with (i) temporary network partitions and (ii) slow processes. What happens? Suggest an adaptation if necessary.

Solution:

(i) In order to deal with temporary network partitions, in each partition, bully algorithm is run locally and each partition chooses its own leader. However, each process keeps running the bully algorithm (continuous heart-beating) and as soon as two partitions merge, the stronger leader (higher process id) bullies the leader of the other partition. This continues until we have only one partition. So, nothing bad happens.

(ii) In order to deal with slow processes, the participants run the bully algorithm, and lower-id-holder processes might, for a while, think that there are no higher-id-holder processes than themselves and declare themselves leaders. However, sooner or later the stronger process's messages will arrive and it will bully the temporary weaker leaders.