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 PAXOS & REPLICATED STATE MACHINES

EXERCISE 1 Basic Paxos

(a) Consider the following system assumptions for running the Paxos algorithm: (1) Each message takes **exactly** 0.5 sec to reach its destination and (2) the processing time at the processes can be ignored (i.e, is zero). In this system, we assume that there are three acceptors (A,B,C), two proposers (D,E) and one learner (F). Proposer D wishes to propose value `"write"` with proposal number 1 as its proposal, and it sends its prepare message at time 0. 0.9 sec later, proposer E, who wants to make its proposal with value `"read"` and proposal number 2, sends its prepare message. During the whole execution, no message gets lost. Draw all messages exchanged during the execution of the Paxos algorithm for this exercise in Figure 1.1, and also specify which value will finally be chosen.

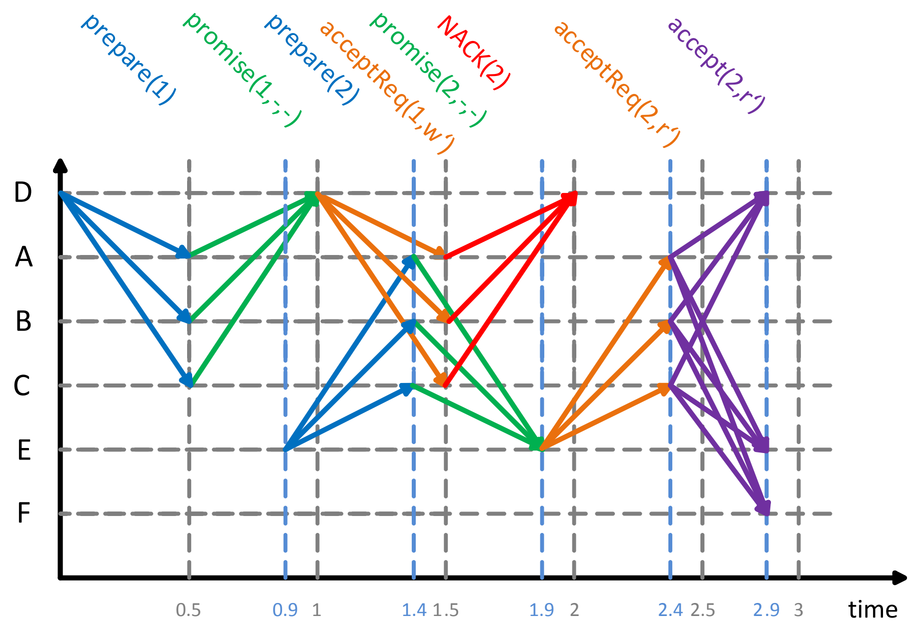
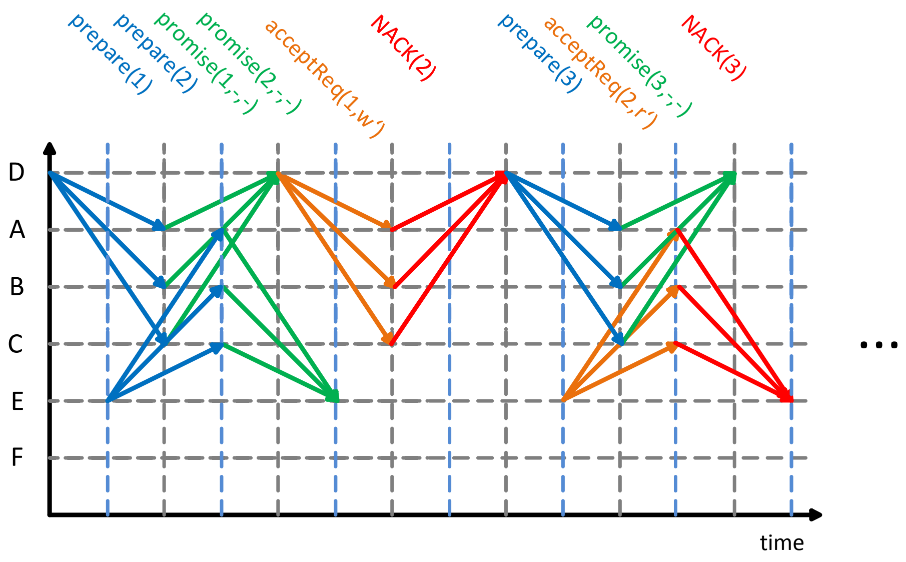


Figure 1.1: **Solution**

(b) Construct a scenario in which the Paxos algorithm does not terminate.

Solution:
A possible scenario would be, for instance, to have two proposers (D and E) who both think they are the leader: Proposer D sends a prepare request with proposal number 1. The prepare request arrives at the majority of the acceptors in the system. Later on, proposer E sends a prepare message with number 2. This message arrives at the majority of the acceptors before the accept request message by D, hence the accept request gets rejected. Imagine this situation continues, i.e., proposer D sends a prepare message with number 3 and then proposer D sends a new accept request, which is again rejected, and so on. In this scenario, liveness is violated and the Paxos instance will never terminate.



EXERCISE 2 RSM Using Paxos

(a) Write a program (in pseudocode) for implementing an RSM using the Paxos algorithm.

Solution:
Idea of solution:

- The values to agree on are the commands
- One instance of the Paxos algorithm per command
- Each process has all three roles (proposer, acceptor, learner)
- A leader, which has to be elected, is the only process that issues proposals (i.e., distinguished proposer and learner). The leader determines the order of commands.

If *n* is the number of commands issued by the client we need *n* instances of the Paxos algorithm.

- Clients send commands to the leader
- Leader proposes each command to a specific Paxos instance ($cmd_i \rightarrow p_i$)
- As cmd_i is the only proposal in p_i every p_i decides on command cmd_i in the end.
- Each server sees the same sequence of need-to-be-executed commands.
- The distinguished learner informs the client that the command has successfully been replicated.

```
init():
       FOR i in {1.. n}           // number of commands
              pi <- new Paxos()   // init new instance
              pi.prepare(i)       // do phase 1 for all instances
              paxoses[i] <- pi    // add instance to an array of instances
       END FOR
       seqNo <- 1               // init sequence number for commands

ON receiveCommand(cmd)
       p <- paxoses[seqNo]      // get the next instance from the list
       p.acceptReq(seqNo, cmd) // send accept request with the given command as value
       seqNo <- seqNo + 1     // increment command sequence number
```

```
}

ON receiveLearn(seqNo, cmd)
       commands[seqNo] <- cmd   //store command
       notifyClient(cmd)        // confirm replication to client
}
```

Listing 2.1: Program at the leader.

```
ON leaderFail()
       runElection()            // determine new distinguished proposer
}

ON receiveLearn(seqNo, cmd)
       commands[seqNo] <- cmd    //store command
}
```

Listing 2.2: Program at the other (non-leader) processes.

(b) Assume that at fictitious time t, two processes think that they are the distinguished process (i.e., proposer & learner). Discuss what happens to the system in terms of the safety and liveness requirements.

Solution:
In the proposed case, the safety requirement is still maintained. However the liveness requirement is broken and the system may not reach consensus until only a single leader is elected. By maintaining property P2c the system is guaranteed to be safe even in this case, since different processes will never disagree on a chosen value.

Property P2c says: For any v and n, if a proposal (i.e., accept request) with value v and proposal number n is issued, then there is a set $\mathcal{S}$ comprised of a majority of acceptors such that either

- no acceptor in $\mathcal{S}$ has ever accepted any proposal numbered less than n
- or, v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in $\mathcal{S}$

However if the system has two distinct leaders, each is going to keep "flooding" the system with different proposals preventing normal operation. This happens because new values may never be agreed on, until only one leader is elected.

(c) Explain what could be done if there are gaps in command execution (cf. Figure 2.2). I.e., a particular instance among the Paxos instances that are realizing the RSM failed to decide for a value (command).

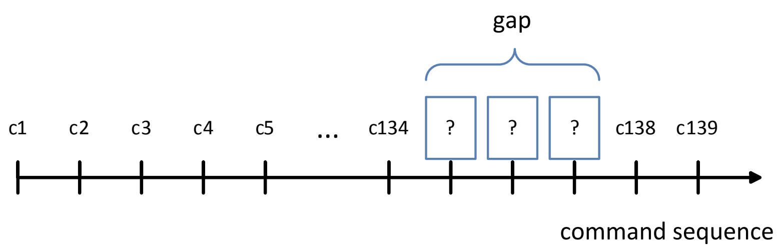


Figure 2.2: Gaps in command sequence.

Solution:
There are two options to fill the gaps created by algorithm.

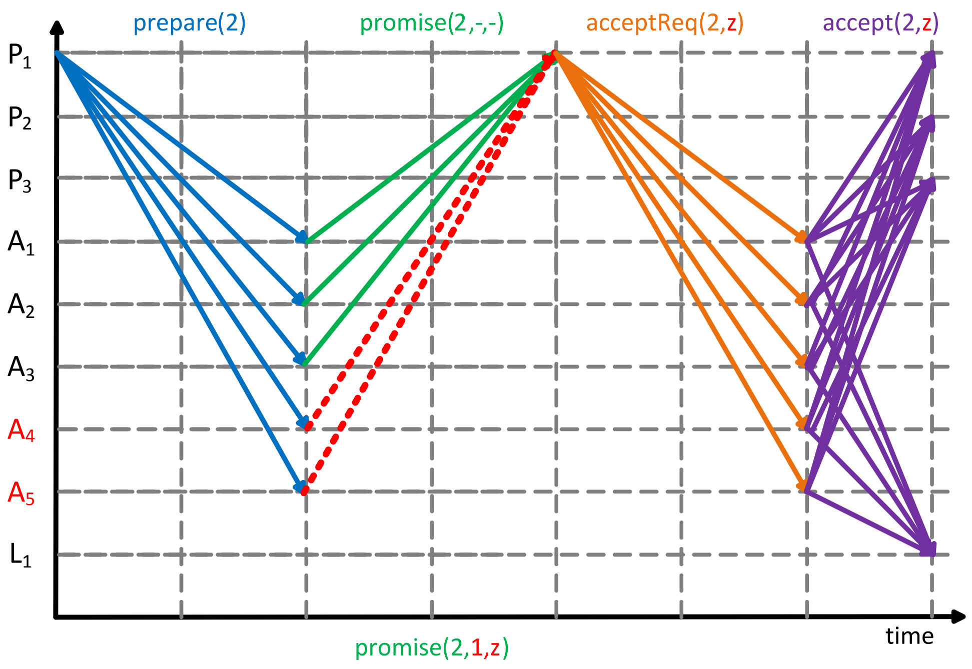
(a) Once the leader determines that no value was decided for, i.e., the command was not successfully replicated, it will re-issue the command in a new `acceptReq` message.

(b) The leader can also propose special *no-op* commands (that leave the state unchaged) by executing phase 2 of the Paxos algorithm. This way, the gaps can be filled and the sequence is completed without changing the order of client commands. The client commands, however, that led to the gaps are lost on all replicas.

EXERCISE 3 Byzantine Paxos

Suppose that we have three proposers, five acceptors and one learner that run the Paxos algorithm. We assume that proposers and the learner never fail, but two out of five acceptors are suffering from arbitrary (byzantine) failure (i.e., they can lie and send wrong messages to others, and generally speaking, they do not follow their own specification). Using a fictitious scenario, show that this Paxos algorithm is no longer satisfying the non-triviality safety requirements of the Paxos specification.

Solution:
Using the following example, we can demonstrate that the non-triviality safety requirement is violated. Suppose that a proposer wants to propose a value v with proposal number 2. Correct acceptors respond to this proposer with message promise(2,-,-). However, the two byzantine acceptors respond to this proposer with message promise(2, old proposal number 1, value z). Then, according to the basic paxos algorithm, the proposer must propose value z instead of its own value, v. Then, the proposer sends an accept request message (2, z), and finally z will be chosen. This violates the safety since z has never been proposed by any proposer before.


prepare(2)
promise(2,-,-)
acceptReq(2,z)
accept(2,z)
P1
P2
P3
A1
A2
A3
A4
A5
L1
time
promise(2,1,z)