Distributed Systems (MS Cybersecurity)
Written test - SOLUTION

First name: _____ Last name:_____ Enrollment num.:_____
Email: _____

Rules: 4 questions that need to be answered in max 120 minutes; no electronic device is allowed during the exam; ask the instructor for paper if you need it. The list of students with a grade above 17 in the written test will be available in the exam page of the website. Students will be identified by enrollment numbers. Only students on that list are admitted at the oral examination. The date of the oral examination is the 19th of February (15:00-17:00), address: Room A6, Via Ariosto 25 (in case of clash with another exam send an email). The full mark (with lode) will be awarded only to students reaching at least 30 in the written test and correctly answering questions during the oral examination.

#Q1: (Pt. 8) (Algorithm Analysis) Consider the following modified code for Lazy Reliable Broadcast.

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
correct := Π;
from[*p*] := $[\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | *m* $\rangle$ **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, *m*] $\rangle$;

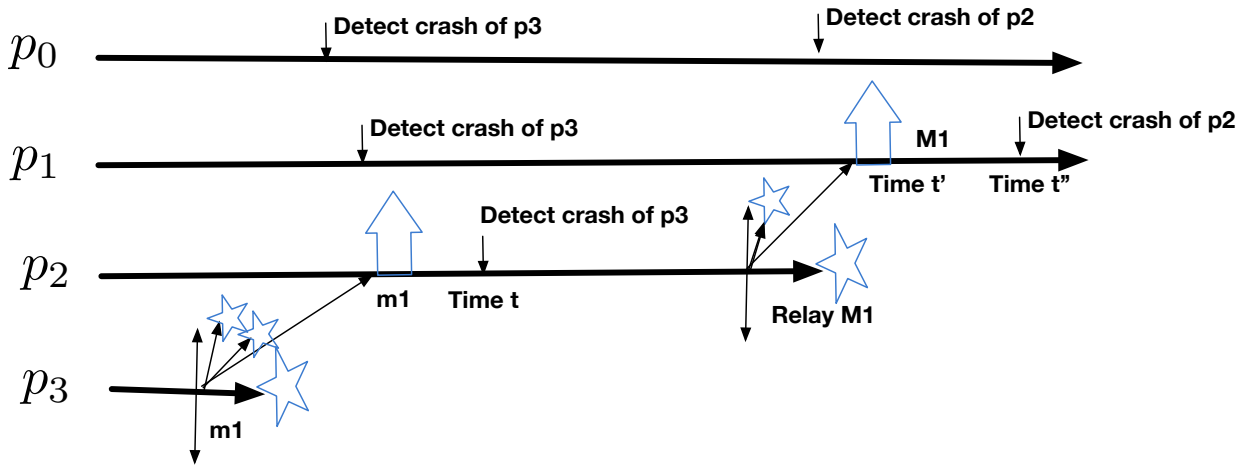
upon event $\langle$ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] $\rangle$ **do**
if $m \notin$ *from*[*s*] **then**
trigger $\langle$ *rb*, *Deliver* | *s*, *m* $\rangle$;
from[*s*] := *from*[*s*] $\cup \{m\}$;
if $p \notin$ *correct* **then**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *s*, *m*] $\rangle$;

upon event $\langle \mathcal{P}$, *Crash* | *p* $\rangle$ **do**
correct := *correct* $\setminus \{p\}$;
forall $m \in$ *from*[*p*] **do**
trigger $\langle$ *beb*, *Broadcast* | [DATA, *p*, *m*] $\rangle$;

Q1.1) Discuss if the property below holds in a system of N processes where an arbitrary number F (with F<N) of processes may crash. If it does not hold show a counter-example that violates it.

- ***Agreement:*** If a message *m* is delivered by some correct process, then any other correct process delivers *m*.

Yes it violates agreement if f>=2 see an example below.



The execution is as follow:

Time 0: p3 broadcasts m1 and then crashes, only p2 will receive it

Time 1: p1 detects the crash of p3, no action apart the removal of p3 from correct is taken. Data,p3,m1]

Time 2: p2 receives m1

Time 3: p2 detects that p3 crashed. The crash handler kicks in and it make p2 relays m2=[Data,p3,m1]

Time 4: p2 crashes only p1 will ever receive the m1.

Time 5: p1 receives [Data,p3,m1] from p2. The BebDeliver handler kicks in, p1 checks in the first if that m1 does not belong to from[p3]. Then it enters in the if, it rbDelivers m1 and updates from[p3]. After it checks the second if in the handler, but at this time p2 the relay process of m1 is not detected as faulty (p2 \in correct for p1). Thus p1 does not execute anything.

Time 6: p1 detects that p2 is crashed. The crash handler kicks, but from[p2] is empty no action is taken.

P0 will never deliver m1, that was delivered by the correct P1. Agreement violated

Q1.2) In case Agreement does not hold for arbitrary values of F, what are the values of F for which the above algorithm is correct?

The algorithm is obviously correct for F=0. But also for F=1. The informal intuition is that to exploit the bug you need a relayer that crashes, if there is only one faulty all relayer are correct.

The formal explanation follows:

Suppose that p crashes while sending message m. And a correct process p' delivers m.

We have that P' delivers m, either before (Case 1) or after (Case 2) it detects that p crashed.

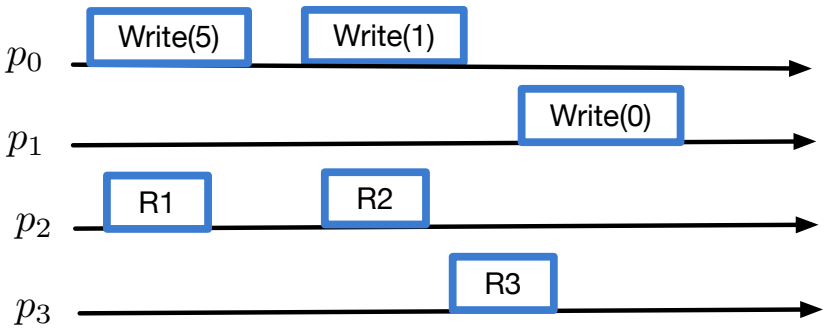
Case 1: P' delivers m before it detects the crash of p. In this case when the crash handler kicks in then P will have that m is in from[p']. So P' BebBroadcast m and p' being correct all corrects receive including P''.

Case 2: P' delivers m after it detects the crash of p. In this case the second if of the BebDeliver n will trigger:

P' received [Data,p,m] from p, but p \not\in correct.

However, the second if makes P' BebBroadcast m and p' being correct all corrects receive including P''

#Q2: (PT. 7) (Registers and consistency) Consider the execution depicted in the following figure and answer the questions:

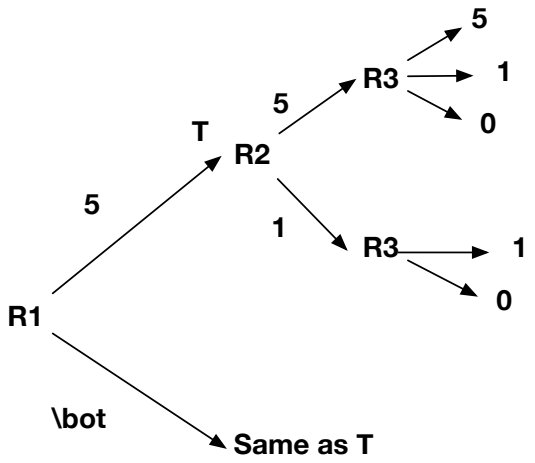


1. Define ALL the values that can be returned by read operations (Rx=?) assuming the run refers to a regular register.

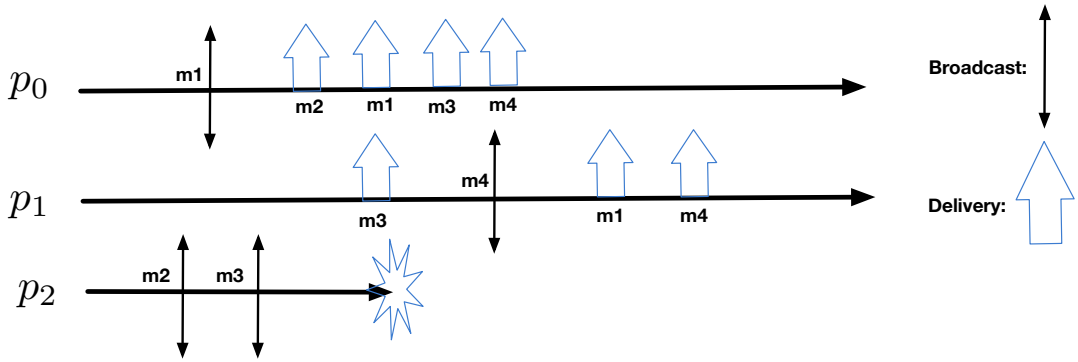
R1={\bot,5} R2={1,5} R3={1,0,5}

2. Define ALL sequences of values that can be returned by read operations (Rx=?) assuming the run refers to an atomic register.

The tree of all possible executions follows:



#Q3: (Pt. 7) (Broadcast) Consider the following run of a broadcast algorithm.



Q3.1: Writes all the happened-before relationships induced by the above run.

The relationships are:

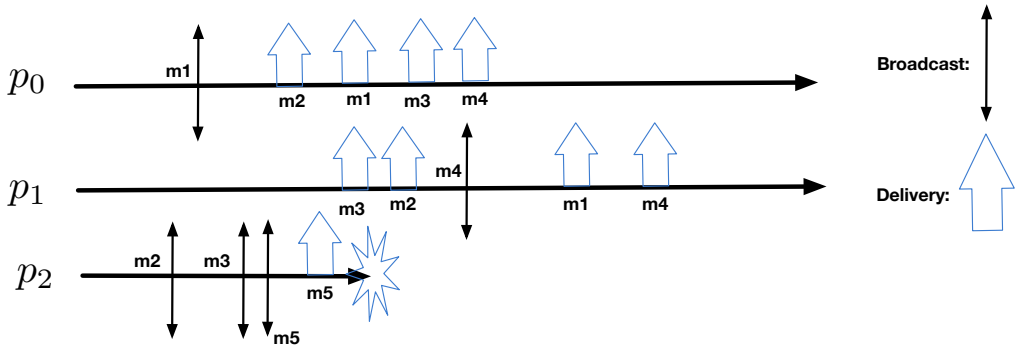
M2 -> M3

M3 -> M4

M2 -> M4

Q3.2: Add the minimal number of broadcast events and the necessary delivery events such that the run satisfies the properties of regular reliable broadcast but not the properties of uniform reliable broadcast.

The only possibility is to create the broadcast event of a message m5 on p2, and to make m5 only delivered by p2. Of course we have to add the delivery of m2 and m3 on p1, otherwise we also violate regular reliable.

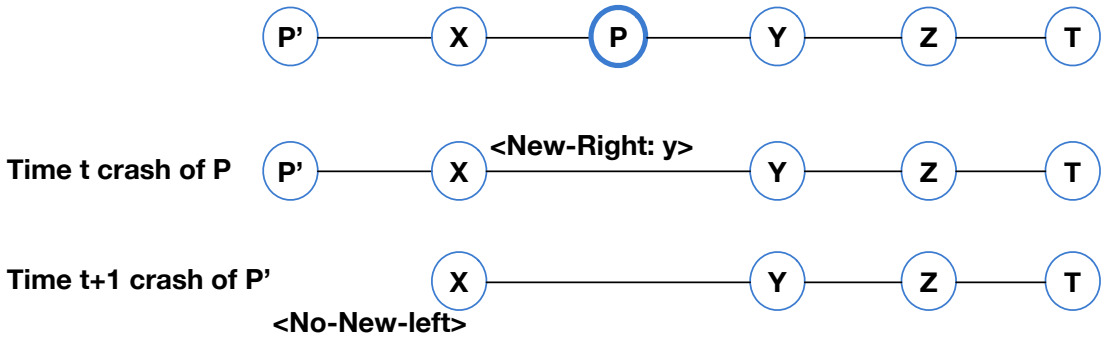


The above run is not uniform. There is a process p2 that delivers a message m5 while all correct processes do not deliver m5. This violates Uniform Agreement.

#Q4: Pt. 10 (Algorithm Design) Consider processes arranged as a line, in a system where each one has a unique ID (see Figure). Processes are connected by perfect point2point links. Processes are not always correct, and when they fail, they fail by crashing. Processes can communicate only with neighbors by exchanging messages.

When a neighbor of a process fails a new neighbor is given: there is an oracle that notifies a process of a new link with a new-left or new-right neighbor. The only exception is if a process becomes an endpoint of the line (P' is an endpoint). In this case the event no-new-left (or no-new-right) is generated.

Suppose that the oracle does not shuffle the line, that is at each reconfiguration the relative order from left to right on the line is preserved (see figure) below.



Q4.1) Create and write the pseudocode of an algorithm that implements a total order broadcast (TOBcast). A TOBcast is a regular reliable broadcast with the additional property that all correct processes deliver messages in the exact same order.

The solution assumes that the link are FIFO.
The idea is to send each message to the left-most process (the left endpoint), that will act as a sequencer relaying messages with a delivery sequence number. All correct process will TOdeliver a received message only if associated with a sequence number (SN) and SN is the number to be delivered. At the beginning number 0 is SN to be delivered. Essentially, the idea is to implement a FIFO for messages relayed by the leader.
A bit of care has to taken to ensure that everything works with failures. Two failures are possible:

- **Failure of a non leader process**: In this case is enough to send again all messages to your left and right neighbor.
- **Failure of a leader process:** In this case we have to elect a new leader. The most clean solution is to elect the new leftmost process as leader (as an example X at time t+1). This ensures a nice property that a message ordered by a defunct leader is either received by the new leader or by no process. This property is fundamental in the correctness of the algorithm.

Below there are two algorithm one that assume FIFO links between processes, one that do not. Each algorithm is an acceptable solution.

Algorithm 18 Line total leader - FIFO LINKS

upon event INIT
  $left = P2pLink()$
  $right = P2pLink()$
  $leader = False$ ▷ True on the leftmost process, trivial check $left = \perp$
  $number\_to\_deliver = 0$
  $leader\_sn = 0$ ▷ Only used by the leader p_l
  $PendingMessages = \emptyset$
  $Ordered = \emptyset$

upon event TO BROADCAST(m)
  $m =< DATA, p_i, m >$
  $PendingMessages = PendingMessages \cup \{m\}$
  SENDLEFT(m)

upon event NEW LEFT NEIGHBOUR(new left p)
  $left = p$
  if $p = \perp$ **then**
    $leader = True$
    $leader\_sn = number\_to\_deliver$
  else
    for all $m \in PendingMessages$ **do**
      sendLeft(m)

upon event NEW RIGH NEIGHBOUR(new right p)
  $left = p$
  for all $m \in PendingMessages$ **do**
    sendRight(m)

upon event DELIVERY FROM PERFECT LINK(MSG)
  if $MSG =< ORD, *, *, * > \wedge leader = False$ **then** ▷ If the message is the ordered kind
    $Ordered = Ordered \cup \{MSG\}$
  $PendingMessages = PendingMessages \cup \{MSG\}$
  sendLeft(MSG)
  sendRight(MSG)

upon event $\exists MSG =< DATA, p_i, m > \in PendingMessages \wedge leader = True \wedge < ORD, p_i, m, * > \notin Ordered$ ▷ This handler triggers when it exists an unordered message, that is a message that is pending but not in the order set. If I am the leader is my job to order it
  $MSG =< ORD, p_i, m, leader\_sn >$
  $leader\_sn = leader\_sn + 1$
  $Ordered = Ordered \cup \{MSG\}$
  $sendRight(MSG)$

upon event $\exists MSG =< ORD, p_i, m, sn > \in Ordered \wedge sn = number\_to\_deliver$
  $number\_to\_deliver = number\_to\_deliver + 1$
  TRIGGER TO DELIVER(p_i, m)

The solution above assume FIFO point to point link

Algorithm 19 Line total leader - NOT ASSUMING FIFO LINKS

upon event INIT
  $left = P2pLink()$
  $right = P2pLink()$
  $leader = False$ ▷ True on the leftmost process, trivial check $left = \perp$
  $number\_to\_deliver = 0$
  $leader\_sn = 0$ ▷ Only used by the leader p_l
  $PendingMessages = \emptyset$
  $Ordered = \emptyset$

upon event TO BROADCAST(m)
  $m =< DATA, p_i, m >$
  $PendingMessages = PendingMessages \cup \{m\}$
  SENDLEFT(m)

upon event NEW LEFT NEIGHBOUR(new left p)
  $left = p$
  if $p = \perp$ **then**
    $leader = True$
    $leader\_sn = number\_to\_deliver$
  else
    for all $m \in PendingMessages$ **do**
      sendLeft(m)

upon event NEW RIGH NEIGHBOUR(new right p)
  $left = p$
  for all $m \in PendingMessages$ **do**
    sendRight(m)

upon event DELIVERY FROM PERFECT LINK(MSG)
  if $MSG =< ORD, *, *, * > \wedge leader = False$ **then** ▷ If the message is the ordered kind
    $Ordered = Ordered \cup \{MSG\}$
  $PendingMessages = PendingMessages \cup \{MSG\}$
  sendLeft(MSG)
  sendRight(MSG)

upon event $\exists MSG =< DATA, p_i, m > \in PendingMessages \wedge leader = True \wedge < ORD, p_i, m, * > \notin Ordered$ ▷ This handler triggers when it exists an unordered message, that is a message that is pending but not in the order set. If I am the leader is my job to order it
  while $\exists$ a message in $Ordered$ with serial number equal to $leader\_sn$ **do**
    $leader\_sn++$
  $MSG =< ORD, p_i, m, leader\_sn >$
  $leader\_sn = leader\_sn + 1$
  $Ordered = Ordered \cup \{MSG\}$
  $sendRight(MSG)$

upon event $\exists MSG =< ORD, p_i, m, sn > \in Ordered \wedge sn = number\_to\_deliver$
  $number\_to\_deliver = number\_to\_deliver + 1$
  TRIGGER TO DELIVER(p_i, m)

The solution above does not assume FIFO point to point link.

Q4.2) Discuss why in your algorithm is not possible for any two correct process to deliver any two messages in different order. What happens, in your algorithm, when we consider any pair of processes? Do they deliver messages in the same order? Discuss with examples.

The proposed algorithm does not ensure the same ordering of messages on correct and faulty processes.
As an example if the first half of the line fails while ordered messages are propagating from the dying leader, then this half will deliver some message in a certain order.
The remaining half of the line will elect a new leader that has not seen all the messages in the dying line, and it could differently order a set of message that will be delivered by the correct processes in the remaining half.