efrei PARIS
École d'Ingénieurs du Numérique

Prénom BOUKHELOUA
Nom Rémi
Promotion Solo
Groupe

20150257: BOUKHELOUA Rémi
M1:
ST2DISC-DE (11/01/2019)
Amphi rouge

e cadre

V

Promotion M1 Big Data

Module Distributed Algorithms

Code cours ST2DISC-CM/1819S7

17

DE - 1h45 min

11/01/2019 16 :00-17 :45

Sujet proposé par : Petr Kuznetsov

Calculatrice autorisée : ☐ **OUI** ☒ **NON**

Documents autorisés : ☐ **OUI** ☒ **NON** **Type de documents : tous**

Ordinateur portable autorisé : ☐ **OUI** ☒ **NON**

Internet : ☐ **OUI** ☒ **NON**

Traducteur électronique, dictionnaire : ☐ **OUI** ☒ **NON**

Consigne :

Merci de restituer uniquement : **le sujet**

Rappel :

- Tous les appareils électroniques (téléphones portables, ordinateurs, tablettes, montres connectées ...) doivent être éteints et rangés.
- Il est interdit de communiquer.
- Toute fraude ou tentative de fraude fera l'objet d'un rapport de la part du surveillant et sera sanctionnée par la note zéro, assortie d'une convocation devant le conseil de discipline. Aucune contestation ne sera possible. Tous les documents et supports utilisés frauduleusement devront être remis au surveillant.
- Aucune sortie de la salle d'examen ne sera autorisée avant la moitié de la durée de l'épreuve.

« Code cours » - « Intitulé du cours » - « type de l'épreuve » **Date de l'épreuve**

Ne rien inscrire dans ce cadre

(4) For trying

Problem 4 (6 points)

A weak-counter object stores a natural value (initially 0) and exports two operations: *inc*, which adds 1 to the counter value, and *read*, which returns the counter value.

A counter object exports one operation *inc-read*, which increments the counter and returns the old value.

4.1 Give a wait-free linearizable implementation of weak-counter using an atomic snapshot object.

4.2 Show that counter has consensus number 2.

4.1. Wait free atomic snapshot ~~object~~ exports two operation snapshot() and write().

function snapshot() :

[x_1 ... x_n] = scan(R_1, ..., R_n)
while true do
  [y_1, ..., y_n] = [x_1 ... x_n]
  [x_1, ..., x_n] = scan(R_1, ..., R_n)
  if [x_1 ... x_n] = [y_1 ... y_n]
    return [x_1 ... x_n]
  else if moved_j and x_j ≠ y_j
    let x_j = (u, U)
    return U
for each moved_j ≠ moved_i V x_j ≠ y_j

function write(v) by p_i:
  S = snapshot()
  R_i.write(v, S).

The idea ~~is to turn this implementation into~~ is to modify write into an inc function that increment the counter instead of replacing old value by v. So, I would do :

~~write~~ inc() by p_i :
  S = snapshot()
  R_i.write(x_i.inc-read() + 1, S).

read?

function inc() by p_i :
  S = snapshot()
  R.write(x_i.inc-read() + 1, S).

It will ~~write~~ read its old value and add 1 to it and then ~~propose~~ replace old by this one as initial implementation did.

~~Yes~~ The problem is that we will face ~~same problems as before~~ with such an implementation due to the fact that As every process manages its own R_i, we should not have issues of concurrency and as it's a wait free algorithm, we are sure that every process will make progress.

Very close...

4.2. The Snapshot.

BOURHELOUA
Rémi

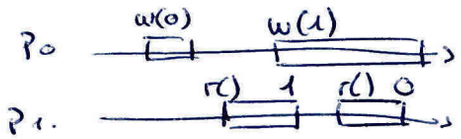
6

Big Data.

Problem 1 (6 points)

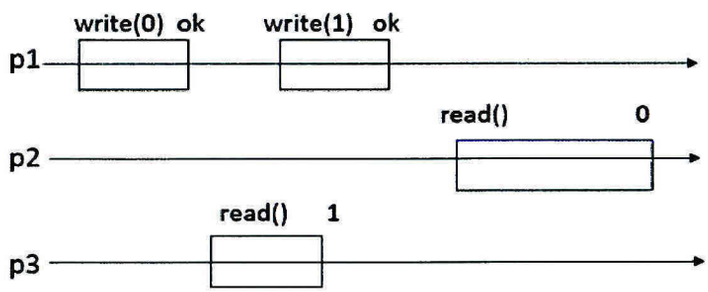
- Depict a history of a one-writer one-reader register that satisfies the specification of a regular register, but *does not* satisfy the specification of an atomic register.

A regular register is register where read() operation return last written value ~~or register value of concurrent write~~ if no write() perform at same time. In the case write() performs at same time, it returns last written value or concurrently written value. Meanwhile an atomic register ~~should be like~~ produces history that have to be linearizable.
A regular register not satisfying atomic specification would be:



+

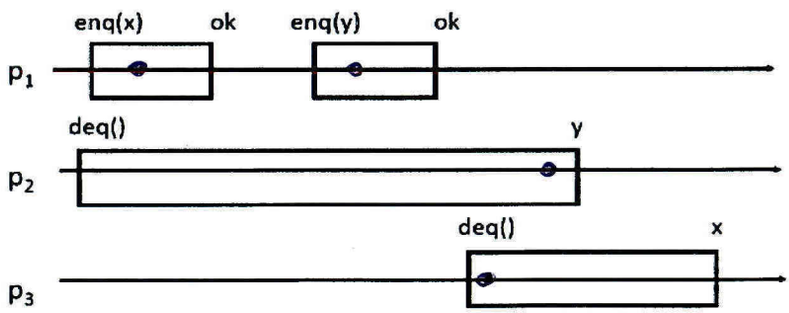
- Is this a history of a regular register (Yes/ No)? Why?



+

As the write(1) is finished when read() of p2 starts, ~~the~~ at that moment the last written value is 1. So read()→0 is not returning the last written value, this is not a history of a regular register.

- Is the history below linearizable with respect to the specification of queue? (Yes/ No) If yes, assign a linearization point to each operation.



+

Yes it is a linearizable history

(4)

Problem 2 (4 points)

We say that a property P *is stronger than* a property P' if $P \subseteq P'$. What is the relation between *deadlock-freedom* ("if every process is correct, then at least one process makes progress") and *obstruction-freedom* ("every process that runs in isolation for sufficiently long makes progress")? Explain why.

Deadlock freedom (DF) and obstruction freedom (OF) are both dependant liveness properties but DF is blocking property meanwhile OF is a non blocking one. So, the first one requires every process to be correct (not faulty) ~~to guarantee~~ whereas the second "only" needs every process to run in isolation. So their condition of success is different and not comparable. ~~Because of that, we can't establish~~

As a property P is stronger that a property P' if all its run are included in all P' runs, if we can't ensure that neither all ~~property of~~ runs of P are in P' and all ~~proper~~ runs of P' are in P ones, we can't conclude on a relation between P and P'.

Because condition of DF and OF are not comparable, we can't ensure that DF runs are completly included in OF ones (and opposite). As a result, we can't conclude ~~on a~~ relation between OF and DF (as we did between OF and Starvation f. on class).

* that there is no

+

(2) For trying

Problem 3 (4 points)

Consider a t-resilient read-write shared memory system ($0 < t < N$) with *initial* failures only: faulty processes take no steps. Give a consensus algorithm in this system. Explain why the proof of the consensus impossibility does not work in this case.

A 0-resilient implementation of a read write shared memory is with p_0 deciding

upon propose(v) by p_i:
  if $i = 0$ D.write(v)
  wait until D.read() $\neq \perp$
  return v.

if $i = 0$, then request from p_0 so we write
else we wait until p_0 ~~crashed~~ exits.

not really)

p_0 can be initially faulty

~~Consensus impossibility proof is based on the fact that for $N = 2$,~~
~~if p_0 proposes 0 and p_1 proposes 1, we won't have~~
~~a majority to select.~~

Consensus is based on three properties, agreement (~~every process agrees on one value~~ no 2 processes can have different values), validity (no value can be elected if not proposed by one process at least) and termination (only correct process proposes). Usually, to demonstrate consensus impossibility, we state that if at one moment remains only 2 processes and both processes votes for different values (0 and 1 for example) we can't find out a majority. ~~and we are done~~ ???

But ~~the~~ here, we have all failures at the beginning, so remains only correct processes (for ~~a~~ for ever).

yes, so?