Concurrent Algorithms November 23, 2020

Solutions to Exercise 8

Problem 1. The following algorithm solves the problem:

uses: C_0, C_1 – counters

upon *propose*(v) **do**
  while *true* **do**
    $(x_0, x_1) \leftarrow$ *readCounters*()
    if $x_0 > x_1$ **then** $v \leftarrow 0$
    else if $x_1 > x_0$ **then** $v \leftarrow 1$
    if $|x_0 - x_1| \geq n$ **then return** v
    C_v.*inc*()

The *readCounters* procedure atomically reads both counters C_0 and C_1. It can be implemented as follows:

upon *readCounters*() **do**
  while *true* **do**
    $x_0 \leftarrow C_0$.*read*()
    $x_1 \leftarrow C_1$.*read*()
    $x_0' \leftarrow C_0$.*read*()
    if $x_0 = x_0'$ **then return** (x_0, x_1)

Problem 2. The answer is yes. To justify this, we show linearizability and termination still hold. For linearizability, we need only to justify the return value of the replaced condition. Consider the first scan s which returns on this condition. (The "first" scan refers to when the scan starts.) Since the timestamp τ of the snapshot ret returned by s is no less than ts (which is obtained at the beginning of s), therefore the $wInc$ procedure which returns τ (denoted by $wInc_1$) cannot end before the $wInc$ procedure which returns ts (denoted by $wInc_2$) starts, by the property of the weak counter. In other words, $wInc_1$ ends no earlier than $wInc_2$ starts. Thus the call of scan (denoted by s_{ret}) inside the update which writes ret ends no earlier than s starts. I.e., two scans s and s_{ret} are concurrent. As a result, s can be linearized at the same point as s_{ret}. Since s_{ret} returns a linearizable value, then s also returns a linearizable value. We can extend the reasoning to infinity by induction. For termination, it is easy to see that now the implementation has more chances to return, and therefore must satisfy termination (as the original implementation satisfies termination).