CSC_4SL05_TP/SLR206B: project pseudocode

Our multi-writer multi-reader atomic register implementation (a variant of ABD [1]) is described in Algorithm 1.

To write a new value v, a writer increments its local sequence number r, sends a *read request* $[?, r]$ to all the processes, waits for the responses of the type $[v', t', r]$ from *a majority* ($> n/2$) of the processes (we also say a *quorum*). Then it picks up the highest timestamp t_{max} in the set of received responses, computes its new timestamp as $t = t_{max} + 1$ and sends a *write request* $[v, t]$ to all the processes in the system. The operation completes as soon as responses of the type $[ack, v, t]$ are received from a quorum.

To read the register value, a reader increments its sequence number r and sends a *read request* $[?, r]$ to every all the processes. As soon as responses of the type $[v', t', r]$ are received from a quorum, the operation selects the value v_m equipped with the highest timestamp t_m. If there are multiple different values written with timestamp t_m, the largest such value is chosen. To make sure that subsequent read operations will not "miss" the returned value, the reader then sends a write request $[v_m, t_m]$, waits until a quorum confirms it by sending $[ack, v_m, t_m]$, and only then returns v.

Notice that, as multiple values can be written with the same timestamp, the value is attached to the acknowledgement sent to a write request. This is needed to ensure that the acknowledgement is indeed sent to *this* write request and *not* to an earlier write request with the same timestamp but a different value (this could happen if the operation is concurrent with multiple writes using the same timestamp. An alternative solution is to maintain an additional sequence number attached to every write request a process issues.

References

[1] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message passing systems. *J. ACM*, 42(2):124–142, Jan. 1995.

Algorithm 1 Multi-writer multi-reader atomic register: pseudocode

1: **Local variables:**
2: $localValue = 0$ // locally stored value, initially 0
3: $localTS = 0$ // locally stored timestamp, initially 0
4: $t = 0$ // timestamp
5: $r = 0$ // sequence number (the number of issued read requests)

6: **Upon** ***write***(v)**:**
7: $r++$
8: send $[?, r]$ to all // read request
9: wait until received $\{[v', t', r]\}$ from a majority
10: let t' be the highest received timestamp
11: $t = t' + 1$ // new timestamp
12: send $[v, t]$ to all // write request
13: wait until received $[ack, v, t]$ from a majority
14: return *ok*

15: **Upon** ***read***(v)**:**
16: $r++$
17: send $[?, r]$ to all // read request
18: wait until received $\{[v', t', r]\}$ from a majority
19: let v_m be the largest value with the highest timestamp t_m
20: send $[v_m, t_m]$ to all // write request
21: wait until received $[ack, v_m, t_m]$ from a majority
22: return v_m

23: **Upon received** $[v', t']$ **from** p_j**:**
24: **if** $t' >$ *localTS* or ($t' =$ *localTS* and $v' >$ *localValue*) **then**
25: *localValue* $:= v'$
26: *localTS* $:= t'$
27: send $[ack, v', t']$ to p_j

28: **Upon received** $[?, r']$ **from** p_j**:**
29: send $[localValue, localTS, r']$ to p_j