Assignment 6

Claim 1. *Algorithm 1 is a linearizable, obstruction-free implementation of a multi-writer snapshot object. Here R is a multi-writer register and S is an array of m multi-writer registers. The multi-writer snapshot object described in Algorithm 1 is not non-blocking.*

Algorithm 1 Operations for the multi-writer snapshot object.

UPDATE(j, v) by process p_i:
$R \leftarrow$ WRITE(i)
$S[j] \leftarrow$ WRITE(v)
return

SCAN() by processor p_i:
do
    $R \leftarrow$ WRITE(i)
    $c \leftarrow$ COLLECT(S)
while READ$(R) \neq i$
return c

Proof. Lemma 2 shows a linearization of the update and scan operations of Algorithm 1. Lemma 3 shows that the multi-writer snapshot object is obstruction-free. Lemma 4 shows that the object is *not* non-blocking by presenting a configuration in which no process can finish its operation in a finite number its own steps. □

Lemma 2. *The implementation of a multi-writer snapshot object in Algorithm 1 is linearizable.*

Proof. Let s be the schedule of a set of operations on the multi-writer snapshot object. Since R is a multi-writer register, the sequence of writes to R is linearizable. Let π_R be the linearization of the low-level write operations to R in the execution of s. Our linearization π of s sets the linearization point of each update and scan operation at the point of their last write operation to R. Note that the linearization point of every operation occurs within its interval of invocation and response.

We consider the following cases to show that π preserves the order of UPDATE and SCAN operations. Let σ_1 and σ_2 be two operations such that σ_1 ended before σ_2 began and let $\pi(\sigma_1)$ and $\pi(\sigma_2)$ be their respective linearization points under π.

1. Let σ_1 and σ_2 both be UPDATE operations. Suppose for a contradiction that $\pi(\sigma_1) > \pi(\sigma_2)$. The write-to-$R$ of σ_1 must have taken place after the write-to-R of σ_2. This is a contradiction since σ_1 terminated before σ_2 (and by extension the write-to-R of σ_2) began.

2. Suppose σ_1 and σ_2 are both SCAN operations. If $\pi(\sigma_1) > \pi(\sigma_2)$ then the last write of σ_2 occurs before the last write-to-R of σ_1. This is impossible as σ_1 ended before the start of σ_2.

3. Suppose σ_1 is an UPDATE operation while σ_2 is a SCAN operation. Suppose for a contradiction that $\pi(\sigma_1) > \pi(\sigma_2)$. Again, this order of the linearization points implies a non-empty intersection between the execution intervals of σ_1 and σ_2. This is a contradiction since σ_1 terminated before the start of σ_2.

4. If σ_1 is as SCAN operation while σ_2 is an UPDATE operation we still cannot have $\pi(\sigma_1) > \pi(\sigma_2)$ by the same reasoning as the above.

□

Lemma 3. *The implementation of a multi-writer snapshot object in Algorithm 1 is obstruction-free.*

Proof. It is easy to see that the UPDATE operation is obstruction-free since it only consists of two write operations to obstruction-free multi-writer registers. Next consider the solo-execution of some process p_i during a SCAN operation. p_i writes i to R then performs a COLLECT operation on S and stores the result in c. Then p_i reads register R. Since no other processor has written to R, the value in R is still i. Thus p_i exists the while-loop and terminates after returning c. □

Lemma 4. *The implementation of a multi-writer snapshot object in Algorithm 1 is not non-blocking.*

Proof. Let p_1 and p_2 be two processes sharing such a multi-writer snapshot object. The initial value in R is 0 and the initial value in S is $\langle 0, 0\rangle$. Suppose both p_1 and p_2 initiate a SCAN operation and consider the following sequence of low-level READ and WRITE operations where op_i is an operation performed by process p_i (ignore the COLLECT operations since they do not interact with R).

$$\text{WRITE}_1(1), \text{WRITE}_2(2), \text{READ}_1().$$

Since p_2 was the most recent process to write to R, $\text{READ}_1()$ will return 2 and p_1 will perform another iteration of the while-loop. Then, if we have

$$\text{WRITE}_1(1), \text{READ}_2(),$$

$\text{READ}_2()$ will return 1 since p_1 was the most recent process to write to R so p_2 will also perform another iteration of the while-loop. The sequence can be extended indefinitely by repeating

$$\text{WRITE}_2(2), \text{READ}_1(), \text{WRITE}_1(1), \text{READ}_2().$$

Neither process can finish within a finite number of its own steps so the multi-writer snapshot object described in Algorithm 1 is not non-blocking. □