Problem 1 (2 points)

Solution

The implementations can be found in the lecture slides (11-13).
Problem 2 (2 points)

Solution. Consider an execution given in the figure below. In the execution, the scan of $p_2$ records the snapshot that does not observe the concurrent update of $p_1$. Process $p_3$ performs a scan that starts after the update of $p_1$ is done, so it has to observe its effects. It performs one collect before $p_2$ writes the results of its scan into its position in the snapshot object and another one after. Because the timestamps of these two elements of the snapshot differ by one, it returns the scan of $p_2$. The scan does not include the update of $p_1$, thus violating atomicity.

Figure 1: An execution violating atomicity
Problem 3  (3 points)

Solution

There are two correct answers.

Yes.

In a system of two or fewer processes, fetch-and-increment can implement an atomic log. We know from class that fetch-and-increment has consensus number 2, thus can be used to implement a universal construction in a system of 2 processes. That universal construction can then be used to implement the atomic log.

No.

In a system of 3 or more processes, there is no wait-free atomic implementation of a shared log from fetch-and-increment objects and registers.

The log object can be used to solve consensus in a system of \(n\) processes, where \(n\) can be arbitrarily large. To do so, upon invocation of \(propose(v)\), process \(p\) simply appends \(v\) to the shared log, then retrieves the log using \(get\log()\) and decides on the first value in the log.

Thus, the log object has consensus number \(\infty\).

If it were possible to produce an implementation \(I\) of a linearizable and wait-free log object using atomic read-write registers and fetch-and-increment objects, then \(I\) could then be used to solve consensus for more than 2 processes. This contradicts the fact that fetch-and-increment has consensus number 2.
Problem 4  (3 points)

Solution

Consider a process \( p \) which is the only process taking steps. Because \( p \) is the only process taking steps and the value of \( C_v \) is incremented in the while loop, then eventually the value of \( C_v \) is going to be greater than the value of \( C_{v-1} \). Therefore, process \( p \) will eventually decide a value, and consequently the algorithm satisfies obstruction-freedom.

The algorithm satisfies validity because if all processes propose the same value \( v \) (which could be either 0 or 1), then they increment the same counter \( C_v \), and consequently the value of \( C_v \) is would be greater than the value of \( C_{v-1} \).

Since the algorithm satisfies obstruction-freedom and validity, then the only property it violates is agreement. To show that it violates agreement consider the following execution in which process \( p_0 \) proposes value 0 and process \( p_1 \) proposes value 1:

- First, process \( p_0 \) executes alone until line 10 and pauses immediately before executing line 10,
- then only process \( p_1 \) takes steps, executing the first iteration of the while loop, where it increments \( C_1 \), and the second iteration of the loop, in which it decides 1 (because \( C_1 = 1 \) and \( C_0 = 0 \)),
- then process \( p_0 \) resumes taking steps incrementing \( C_0 \) at line 10 and executing the second iteration of the loop.
- Because during the second iteration of the loop by \( p_0 \) the values of \( C_0 \) and \( C_1 \) are the same (\( C_1 = 1 \) and \( C_0 = 1 \)), then \( p_0 \) increments \( C_0 \) again and executes the third iteration of the loop in which it decides 0 (because \( C_1 = 1 \) and \( C_0 = 2 \)).