CSE 6117 Distributed Computing Assignment 6

Omar Abid
Student ID: 211295573
CSE Login: omarabid

16-03-2016

1 Problem

We have an asynchronous share memory system with n processes up to f halting failures. Initial states of all shared objects may be defined by the algorithm and we wish to solve the consensus problem satisfying the following conditions.

- **Termination:** in every execution where at most f processes experience halting failures, all non faulty processes eventually output a value.
- **Agreement:** in every execution, all outputs produced in the execution are identical.
- **Validity:** in every exection, each output is the input of some process in that execution.

2 Give a consensus algorithm with one shared stack, and shared read-write registors for $f = 1$

Assume that we have an initial state of the stack with (n-2) elements which have some value representing *loser*, and three additional stack elements (as the last three elements) with some value representing *winner*. Also, initially, each of the registers have a *loser* value indicating that nothing has been written to them yet. We additionally assume that this initial value is not in the domain of values that any algorithm can get as an input value (so as to identify a value written by some process distinguishable from the initial state). We have $n + 1$ stack elements in total. We allow the system to have *mwmr* registers.

Algorithm

Popping from the stack: Each process i pops a value from a stack, and will receive either the element *loser* or *winner*, in the case when no elements are present, the process will see an empty stack.

Case 1: Process receives the value *loser* Then the process continues to read its register R_i until a *loser* initial value is replaced by some other value by some other process P_j. When the process sees that this occurs, outputs that value and terminates.

Case 2: Process receives the value *winner* Then, at this stage we know that at least $n - 2$ processes have popped an element from the stack each of which have some v_k in R_k indicating they are a loser ($1 \leq k \leq n$) and that three elements are remaining: three *winner* elements. Now we write our value v_i onto R_i. Then we attempt to pop another value, and have two more cases.

- Next Popped Element is another *winner*
  - Then we have made a decision of which registers value all other processes will take as their output.
  - This process writes to all other processes in a *mwmr* shared memory system.
- Next Popped Element is *empty*

- Takes the value of the register that popped a second winner element. Since all other processes have their initial register value unchanged as *loser*, this process can identify which process is the *winner* after the second pop by reading and identifying the register other than itself which has some value (other than *loser*) written to it.
- Writes this value to all other registers. Note that there are potentially two processes writing the same value to all other registers, this redundancy is necessary to account for a halting failure as we will see.

Accounting $f = 1$ halting failures.

The algorithm has been described above, but now we will introduce the consequence of one halting failure occurring on solving consensus.

Case I

Failure of process occurs before it has popped a value or immediately after popping one *loser* element from the stack Then the remaining $(n-1)$ processes will continue to pop elements and in the case when a process fails before it pops any value, we have only one element that *pops* two *winner* values. Since we know that no other process (including this winner process) will fail, this process can write its value to every other process. All *loser* elements will output the value of their register when they see it has been updated. In the case when it fails after it pops a *loser* element, then the algorithm will still work since subsequent executions will be indistinguishable between this process failing or surviving (some other process will write to this register, but this process will not output its value since it has halted).

Case II

Failure of process occurring after it pops its first *winner* but before it writes This case is trivial, since it hasn't popped the second *winner* element, some other process will pop and be designated as the "leader", this process will then write its value to all other registers which the remaining surviving processes will output.

Case III

Failure of process occurring after it pops its first *winner* and after it writes Similar reasoning follows as above, although it has written a value, some other process will pop a second *winner* element and will be designated as the "leader". It will then follow the algorithm and write its value to every other register.

Case IV

Failure of process occurring immediately after it pops its second winner element Then the remaining process will attempt to pop an element and see "empty", and seeing this the algorithm will take the value written to the register R_i by the failing process P_i and distribute it to all other registers (including its own). Since the value was written to the register after the first pop, there is a guarantee that at this point, some value is present in that register (that has now halted).

Case V

Failure of process occurring sometime after it pops its second winner element. This is similar to the above case, since we have two processes that write to all other processes, then we have a fail safe mechanism since we know that at most one process fails; that is if this process fails, the second process (which has one *winner* element), will continue writing to all other registers the value R_i of the now halted process P_i. Note that as mentioned earlier, there are two processes that perform write operations to all other processes as a fail safe redundancy mechanism to deal with a halting failure.

Satisfying Conditions

Termination: In the case where no halting failures occur, eventually you have two processes which take two and one winner element respectively. After writing to all other processes these terminate and the loser processes will terminate after reading that their value has changed and outputting this new value written. In the case when up to $f = 1$ halting failures occur, we have considered each of the different cases above and shown that regardless of where the failure occurs, a consensus is reachable and that each process will eventually output a value (even if it takes an arbitrary long period of time).

Agreement: Each of the *loser* processes can only output their value once some winner process has written to it. Each of the winner processes writes the value v_i written to R_i by the i^{th} process which has popped two *winner* elements from the stack. We have shown that these two processes can distinguish an execution where one or two elements have popped so only a single decision of the output value can be made. Hence even if one of these *winner* processes fail, the second will compensate and write the same value to every other process. Hence, all outputs values are identical.

Validity: This is somewhat trivial, once a consensus of the value to be outputted has been made by the *winner(s)* element(s) (depending on if or where in the execution some process fails) this will be the output of these *winner* processes. Additionally, these winner processes write to every other register, hence each of the *loser* processes will have an input which was an output of some process.