Solution of Problem Set 4

November 7, 2005

Question 1(16.14):

The essential reason that this fault-tolerant synchronizer won't work is that the system has no way to distinguish a dead process from a slow process. It is possible for fast processes to proceed and even terminate without waiting for slow processes while the inputs of these slow processes may be critical.

Consider executing *FloodSet* in conjunction with the synchronizer in a completely connected network G of size n, in which $n > 1$ and $f = 1$. Let value set $V = \{0, 1, v_0\}$, and each process either decides on the unique input of all the process or on the default value v_0. Assume all the processes have input 1 and are very fast, except process 1 has input 0 and is very slow. Then it is possible that all the processes except process 1 have finished *FloodSet* algorithm and reached agreement on value 1 because the synchronizer allows these fast processes proceed to the next round without waiting for process 1. However, when process 1 eventually finishes *FloodSet* algorithm, it will decide on value v_0 rather than 1. Then the agreement fails. Since *FloodSet* algorithm has been proved to be correct in synchronous systems, there must be this fault-tolerent synchronizer to be mistaken.

Question 2(12.5):

If all the process failures happen at the very beginning of computation, there are algorithms to solve the agreement problem. Here we present an algorithm similar to *RMWAgreement* algorithm on Page 388. Since we don't have RMW shared memory, we need further mechanism to implement this algorithm. Let R_{dec} be a special register storing the final decision value, which is initialized to be *unknown*. Implement a mutual exclusion algorithm, such as BurnsME, on all the processes, and put the following code into the

critical region of process i:

if ($R_{dec} == unknown$)

$R_{dec} = i$'s initial input;

decide on i's initial input;

else

decide on R_{dec};

Here we don't need to consider high-level-fairness too much since each process will try to enter the critical region at most once. And because there are no further process failures in the middle way, the mutual exclusion algorithm can be executed successfully. The output of the agreement algorithm is the initial value of the process which first enters the critical region and writes its own value onto R_{dec}. It's easy to see that this algorithm works for both 1-failure termination and wait-free termination.

Question 3(10.8):

For this problem, you can either simply rewrite *Bakery* algorithm to a two-process version or make use of the algorithm from *Economical solutions for the critical section problem in a distributed system* by Gary L. Peterson and Michael J. Fisher. Please refer to Page 297 or the paper for the details of the proof.