Information Processing Letters 17 (1983) 43–46
North-Holland

19 July 1983

# A DISTRIBUTED SOLUTION OF THE DISTRIBUTED TERMINATION PROBLEM *

S.P. RANA
*Computer Centre, Indian Institute of Technology, Hauz Khas, New Delhi-110029, India*

Communicated by W.L. Van der Poel
Received April 1982

A distributed and fully symmetric solution is presented for the distributed termination problem. In contrast to the existing solutions, the above solution does not require a predesignated process to detect termination. The case of asynchronous communications is also discussed.

*Keywords:* Distributed programming, termination detection, decentralized control

## 1. Introduction

A reasonable expectation from a program, say P, is that it should terminate soon after accomplishing the task for which it was written. The termination issue is trivial if P is a sequential program; however, if P is a distributed program, additional effort is needed to ensure the proper termination of P. The termination problem was brought into prominence by Francez [1].

The distributed termination problem is briefly stated as follows. We are given a distributed program P consisting of communicating sequential processes $P_1, P_2, \ldots, P_n$. The necessary condition for a process $P_i$ to terminate is that it satisfies a local predicate $B_i$, $1 \leqslant i \leqslant n$. However, a process can only terminate when all other processes of the program P satisfy their local predicates simultaneously or alternatively we say that a process only terminates when the global termination condition is satisfied.

Note that in a distributed environment the processes only communicate by exchanging messages and that there is no central controller to observe the state of all processes simultaneously. Several algorithms have appeared in the literature to solve the distributed termination problem [2,3,4,5]. All of these works rely upon a predesignated process to detect the global termination. The solution presented in this paper differs from others in the vital aspect that any of the n processes may initiate a detection wave and may possibly succeed in detecting the global termination condition. The resulting protocol is fairly simple, as will be seen in the next section.

Although an arbitrary process may initiate a detection wave, eventually the last processes to satisfy their local predicate would succeed in detecting the termination and thus be responsible for initiating the termination wave. After presenting the solution in Section 2, its correctness and performance is discussed in Section 3. The last section concludes with a mention of similar works and comments on the termination problem in the case of asynchronous communications.

## 2. A distributed solution

The solution presented below is fully distributed and syummetric in the sense that each process follows an identical protocol and that there is

* This work was in part supported by DST Project No. HCS/DST/634/78.

no specialized process in P.

It is required that the clocks of the individual nodes executing the processes are synchronized. The feasibility of such a synchronization is established by Lampson [6].

Each process has the ability to note down the clock time whenever it satisfies its local predicate. Also a process sending a message can time-stamp it, if required by writing the current clock time.

The solution presented here is expressed in CSP like notation and hence a synchronous mode of communication is implicit. The solution can easily be modified for the case of asynchronous communications, as will be pointed out in the concluding section.

Briefly, the essential features of the present solution are the following:

(i) The processes are assumed to be connected by a virtual (or real) hamiltonian cycle and the control communication is always on this cycle in a single direction, either clockwise or anticlockwise. Thus each processor has a successor on the cycle to which it can forward a message and a predecessor from which a message can be received.

The control communication is the communication introduced in order to detect termination and to allow the processes to terminate. In contrast to the control communication we have the basic communications which are incurred among the processes to solve the problem at hand.

(ii) Whenever a process satisfies its local predicate, it notes down the current clock-time and forwards a detection message, with the time-stamp equal to the current clock time and with a counter initialized 1, to its successor on the cycle.

(iii) A process is said to be *active* if it is not satisfying its local predicate, otherwise it is said to be *passive*.

If an active process receives a detection message from its predecessor it simply purges it.

(iv) If a passive process receives a detection message, it compares the time-stamp of the message with the noted time when it last satisfied its local predicate. If the latter time is greater than the time-stamp of the message, the message is purged, otherwise the counter of the message is incremented by 1 and is forwarded to the successor on the cycle.

(v) If a passive process receives a detection message with the value of counter equal to n, the global termination is concluded. The process sends a termination message to the successor and terminates.

The solution is now formally presented below.

Let the given distributed problem be expressed as

$$P :: [P_1 \parallel \cdots \parallel P_n]$$

where, for each $1 \leqslant i \leqslant n$, $P_i :: * [S_i]$.

Let BTIME$_i$ denote the time when $P_i$ last satisfied its local predicate $B_i$. Further, let TIME and COUNT denote the time-stamp and counter respectively on a detection message.

First let us write a modified version of a process $P_i$ to reflect the copying of current clock-time whenever the local predicate is satisfied. The process $P_i$ waits in its top level for receipt of basic messages from other processes. Whenever $P_i$ receives a message the predicate again becomes false.

The modified $P_i$ can be expressed as

```
P_i :: B_i := false;
  * [ S_i'
    □B_i → BTIME_i := CLOCK-TIME
```

where CLOCK-TIME gives the current clock-time of the processor holding the process $P_i$.

When $B_i$ is true, the process $P_i$ can send a detection message to its successor denoted as $P_{i+1}$. Detection messages may also be received by $P_i$ from its predecessor $P_{i-1}$. The actions taken by $P_i$ in the above situations are included in the following version of $P_i$:

```
P_i :: B_i := false;
  * [ S_i'
    □B_i → BTIME_i := CLOCK-TIME;
           TIME := BTIME_i;
           P_{i+1} ! detection-message (TIME, COUNT)
    □P_{i-1} ? detection-message (TIME, COUNT) →
      [¬B_i → purge the message
      □B_i →
         [TIME < BTIME_i → purge the message
         □TIME ⩾ BTIME_i →
            COUNT := COUNT + 1;
            P_{i+1} ! detection-message
              (TIME, COUNT)
            ]
         ]
```

Finally $P_i$ must have a provision to check whether its own detection message reached back to it, in which case $P_i$ must initiate a termination wave and then terminate itself. Also if not terminated, $P_i$ must be prepared to receive a termination message from $P_{i-1}$. Thus the final version of $P_i$ is as follows:

```
$P_i$ :: $B_i$ := false;
  * [$S_i'$
    □$B_i$ → $BTIME_i$ := CLOCK-TIME];
          TIME := $BTIME_i$;
          COUNT := 1;
          $P_{i+1}$ ! detection-message (TIME, COUNT)
    □$P_{i-1}$ ? detection-message (TIME, COUNT) →
      [COUNT = n →
        $P_{i+1}$ ! terminate-message;
          TERMINATE
      □COUNT ≠ n →
        [¬$B_i$ → purge the message
        □$B_i$ →
          [TIME < $BTIME_i$ → purge the message
          □TIME ≥ $BTIME_i$ →
            COUNT := COUNT + 1;
            $P_{i+1}$ ! detection-message
              (TIME, COUNT)
          ]]]
    $P_{i-1}$ ? termination-message →
      $P_{i+1}$ ! termination-message;
        TERMINATE
  ]
```

## 3. Correctness

Two assertions are proved below, in order to show the correctness of the proposed solution in the preceding section.

**Assertion 1.** *If the global termination condition is satisfied, termination will be eventually detected.*

**Proof.** Assume that all processes in P satisfy their local predicates simultaneously at some time instant, say t. Suppose that $P_e$ is the latest process to satisfy its local predicate $B_e$, viz. $P_e$ satisfies $B_e$ at time t.

Consider the detection message of $P_e$, having the time-stamp t. Since all processes have satisfied their local predicates for the last time at or earlier than time t, the detection message of $P_e$ will not be purged by any process. Thus $P_e$ will eventually get its detection message back and the termination will be detected.

**Assertion 2.** *There is no possibility of detecting false termination.*

**Proof.** Detecting false termination implies getting a detection message with the counter value n, even when the global termination condition is not satisfied.

Consider a detection message with time-stamp t forwarded by a process $P_i$. This implies that $P_i$ became passive for the last time at or earlier than time t. Process $P_i$ may again become active if some active process, say $P_j$, initiates basic communication with it.

If $P_j$ has already forwarded the detection message with time-stamp t, then $P_j$ was passive at time t and was made active later by another process. Continuing the chain of argument like this, we find that there is an active process $P_k$ which communicates with a passive process who has already forwarded the detection message with time-stamp t; whereas $P_k$ itself has not received the above detection message so far.

It is clear that $P_k$ will purge the detection message when it reaches him because $P_k$ was active at a time equal to or later than t. Thus a detection message initiated at time t never reaches its originator if there is a single process that has been active after time t. Hence there is no possibility of detecting a false termination.

## 4. Discussion

In the solution presented in this paper a process has to make a simple check to decide whether a detection message is to be purged, forwarded or it indicates the termination. It is difficult to estimate the total number of control messages circulated till the detection of termination, because this depends upon the manner in which the values of $B_i$'s change. However, once a global termination condition is true, it takes only one round to detect the termination. The total number of messages required could be reduced by some refinement, but possibly at the cost of delaying the detection of termination.

It is pointed out that more than one process can detect termination. This happens when few processes become passive for the last time simultaneously. Instead of being a problem, the above situation has an advantage that the termination wave is now propagated in parallel.

In the termination detection approaches described in few earlier works (viz. [1,2,5]), a restriction is imposed on creating new channels. The present paper does not assume such a restriction. Further, the above-mentioned works, besides specifying a predesignated node to detect termination, rely upon the existence of a spanning tree over which control communications are propagated. In order to analyze the performance of the above solutions, one must consider the overhead of creating a spanning tree. In the solution presented here, there is no such overhead. Note that if a hamiltonian cycle can be constructed, out of the existing channels, encompassing all the n processes, then, in the present approach, there is no need to introduce additional channels too. However, such a cycle has to be identified first. A solution quite different from ours is proposed in [4] for the hamiltonian cycle based control communication; however, again a special node is predesignated to initiate all control communications.

A notable feature of the present and all earlier approaches is that the processes are assumed to communicate by using only synchronous commands; similar to that of CSP [7]. If an asynchronous communication is assumed, the known solutions do not work. The simple reason is that the condition 'all processes simultaneously passive' is no longer sufficient to guarantee proper termination.

Two potential approaches to deal with the asynchronous communications case are:

(i) Modify the global termination condition: ensure that all processes satisfy their local predicates at a particular time and no message is pending to be delivered.

(ii) Modify the local predicates: all processes are modified such that an acknowledgement is expected (sent) for each basic communication message sent (received) by a process. Let $B_i$ denote the modified local predicate of $B_i$; then $B_i$ becomes true only when $B_i$ is true and all expected acknowledgements have been received.

By using the latter approach, the solution described in Section 2 is directly applicable to the case of asynchronous communications.

## References

[1] N. Francez, Distributed termination, ACM – TOPLAS 2 (1) (1980) 42–55.

[2] N. Francez and M. Rodeh, Achieving distributed termination without freezing, Tech. Rept. TR-72, IBM Israel Scientific Centre, 1979.

[3] E.W. Dijkstra and C.S. Scholten, Termination detection for diffusing computation, EWD-687, Nuenen.

[4] N. Francez, M. Rodeh and M. Sintzoff, Distributed termination with interval assertions, Tech. Rept. TR # 186, Computer Science Department, TECHNION – Israel Institute of Technology, 1980.

[5] K.M. Chandy and J. Misra, Termination detection of diffusing computations in communicating sequential processes, ACM – TOPLAS (1982) (appeared earlier as Tech. Rept. TR-144, The University of Texas at Austin, 1980).

[6] L. Lamport, Time, clocks and the ordering of events in a distributed system, Comm. ACM 21 (7) (1978) 558–565.

[7] C.A.R. Hoare, Communicating sequential processes, Comm. 21 (8) (1978) 666–677.