Solution of a Problem in Concurrent Programming Control

E. W. Dijkstra

September, 1965
Volume 8, Number 9
pp. 569

This short paper marks the beginning of a long era of interest in expressing the synchronization of concurrent processors using ordinary programming languages. This paper considers the problem of implementing an indivisible operation by mutual exclusion of critical sections of code. Later papers by Dijkstra introduced the concepts of semaphores; one of the motivations of semaphores was reducing the complexity of the programming required to implement mutual exclusion. (A letter to the editor by Don Knuth, published in May 1966, points out that Dijkstra’s solution is susceptible to “starvation” – meaning that a particular processor can fortuitously be forever blocked from entering the critical section. Knuth’s starvation-free solution is even more complex than the solution here.)

—P. J. D.

Solution of a Problem in Concurrent Programming Control

E. W. Dijkstra

Technological University, Eindhoven, The Netherlands

A number of mainly independent sequential-cyclic processes with restricted means of communication with each other can be made in such a way that at any moment one and only one of them is engaged in the “critical section” of its cycle.

Introduction

Given this in a paper is a solution to a problem for which, to the knowledge of the author, has been an open question since at least 1962, irrespective of the solvability. The paper consists of three parts: the problem, the solution, and the proof. Although the setting of the problem might seem somewhat academic at first, the reader trusts that anyone familiar with the logical problems that arise in computer coupling will appreciate the significance of the fact that this problem indeed can be solved.

The Problem

To begin, consider N computers, each engaged in a process which, for our aims, can be regarded as cyclic. In each of the cycles a so-called “critical section” occurs and the computers have to be programmed in such a way that at any moment only one of these N cyclic processes is in its critical section. In order to effectuate this mutual exclusion of critical-section execution the computers can communicate with each other via a common store. Writing a word into or nondestructively reading a word from this store are indivisible operations; i.e., when two or more computers try to communicate (either for reading or for writing) simultaneously with the same common location, these communications will take place one after the other, but in an unknown order.

The solution must satisfy the following requirements:

(a) The solution must be symmetrical between the N computers; as a result we are not allowed to introduce a static priority.

(b) Nothing may be assumed about the relative speeds of the N computers; we may not even assume their speeds to be constant in time.

(c) If any of the computers is stopped well outside its critical section, this is not allowed to lead to potential blocking of the others.

(d) If more than one computer is about to enter its critical section, it must be possible to devise for them such finite speeds, that the decision to determine which one of them will enter its critical section first is postponed until eternity. In other words, constructions in which “After you” “After you” blocking is still possible, although improbable, are not to be regarded as valid solutions.

We begin the challenged reader to stop here for a while and have a try himself, for this seems the only way to get a feeling for the tricky consequences of the fact that each computer can only request one one-way message at a time. And only this will make the reader realize to what extent this problem is far from trivial.

The Solution

The common store consists of:

"Boolean array b, (it N); integer k".

The integer k will satisfy 1 ≤ k ≤ N, b[k] and b[k] will only be set by the kth computer; they will be inspected by the others. It is assumed that all computers are started well outside their critical sections with all Booleans arrays mentioned set to true; the starting value of k is immaterial.

The program for the kth computer (1 ≤ i ≤ N) is:

```
begin c[i] = false;
end
```

The Proof

We start by observing that the solution is safe in the sense that no two computers can be in their critical section simultaneously. The only way to enter its critical section is the performance of the compound statement

```
L[i]: begin c[i] = true;
for j = 1 step 1 until N do
if j ≠ i and not c[j] then go to L[i]
end;
c[i] := true; b[i] := true;
remainder of the cycle in which stopping is allowed;
go to L[i]
```

The second part of the proof must show that no infinite “After you” blocking can occur; i.e., when none of the computers is in its critical section, of the computers (i.e., jumping back to L[i]) at least one—and therefore exactly one—will be allowed to enter its critical section in due time.

If the kth computer is among the looping ones, b[k] will be true and the looping ones will all find k ≠ i. As a result one or more of them will find in L[i] the Boolean b[k] true and therefore one or more will decide to assign “k := i”. After the first assignment “k := i”, b[k] becomes false and no new computers can decide again to assign a new value to k. When all decided assignments to k have been performed, k will point to one of the looping computers and will not change its value for the time being, i.e., until b[k] becomes true, var., until the kth computer has completed its critical section. As soon as the value of k does not change any more, the kth computer will wait (via the compound statement L[i]) until all other c’s are true, but this situation will certainly arise, if not already present, because all other looping ones are forced to set their c true, as they will find k ≠ i. And this, the author believes, completes the proof.
Additional Comments on a Problem in Concurrent Programming Control

Professor Dijkstra's ingenious construction (Solution of a Problem in Concurrent Programming Control, Comm. ACM 6 (Sept. 1963), 608) is not, I think, a solution to a related problem almost identical to the problem he posed. Indeed, his system—unless I am misunderstanding his solution—does not even have a mechanism which would allow two processors to be activated in a controlled manner.

1. I would like to comment on Mr. Dijkstra's solution (Solution of a Problem in Concurrent Programming Control, Comm. ACM 6 (Sept. 1963), 608) for the case of two computers. Mr. Dijkstra's solution is clearly highly ingenious, but I would like to suggest that there are some important issues that need to be addressed.

2. The solution presented by Mr. Dijkstra is based on the idea of using a single control point, which he calls a "critical section." This point is used to synchronize two processors, which are executing concurrently.

3. However, the solution presented by Mr. Dijkstra has some limitations. First, it assumes that the processors are always synchronized, which is not always the case in real-world systems. Second, the solution is not efficient in terms of resource utilization, as it requires a large amount of overhead to manage the synchronization.

4. A better solution might involve using a more fine-grained synchronization mechanism, such as semaphores or monitors, which allow for more precise control over the access to shared resources.

5. In conclusion, while Mr. Dijkstra's solution is an interesting approach to the problem of concurrent programming control, it has some limitations that need to be addressed. A more fine-grained synchronization mechanism could provide a more efficient and practical solution to this problem.