Computer Systems

Further Comments on Dijkstra's Concurrent Programming Control Problem

Murray A. Eisenberg and Michael R. McGuire
The MITRE Corporation

Key Words and Phrases: critical section, concurrent programming control, multiprocessing
CR Categories: 4.32

E.W. Dijkstra [1] presented an algorithm whereby N mainly independent computers, with a common data store as their sole means of communication, could contend for exclusive control of any given resource (storage, I/O, etc.). To use the resource, a computer had to gain access to the "critical section" of the algorithm, within which one and only one computer at a time could be executing.

Dijkstra's algorithm was ingeniously constructed to prevent "after you—after you" blockage of the system while restricting critical section access to one computer at a time. His algorithm, however, allowed the possibility that an individual computer, when contending for access might have to wait indefinitely.

Knuth [2] provided a modification to Dijkstra's algorithm which guaranteed access by an individual contending computer within $2^{N-1} - 1$ turns. This algorithm was further modified by deBruijn [3] to provide access, within $\frac{1}{2} N(N - 1)$ turns. The following algorithm guarantees that a computer need wait no more than $N - 1$ turns.

The common store (within which single reads and writes are undividable operations) consists of:

integer array *control* [1 : N]; **integer** k
where $1 \leq k \leq N$, and each element of "control" is either 0, 1, or 2. All elements of "control" are initially zero; the initial value of k is immaterial.

The program of the ith computer ($1 \leq i \leq N$) is

```
begin integer j;
L0:  control [i] := 1;
L1:  for j := k step 1 until N, 1 step 1 until k do
       begin
         if j = i then goto L2;
         if control [j] ≠ 0 then goto L1
       end;
L2:  control [i] := 2;
     for j := 1 step 1 until N do
       if j ≠ i and control [j] = 2 then goto L0;
L3:  if control [k] ≠ 0 and k ≠ i then goto L0;
L4:  k := i;
     critical section;
L5:  for j := k step 1 until N, 1 step 1 until k do
       if j ≠ k and control [j] ≠ 0 then
         begin
           k := j;
           goto L6
         end;
L6:  control [i] := 0;
L7:  remainder of cycle;
     goto L0;
end
```

Authors' address: The MITRE Corporation, Bedford, MA 01730.

To prove that this algorithm is an acceptable substitute for Knuth's it will be shown that:
(a) There exists a protected area within the algorithm which includes the critical section and within which no two computers can be simultaneously processing.
(b) The system cannot be blocked (i.e. when one or more computers are contending for entrance to their critical section, the decision to determine that some computer may enter may not be postponed indefinitely).
(c) An individual computer cannot be blocked (i.e. the decision to determine that an individual contending computer may enter its critical section may not be postponed indefinitely).

Proof. First observe that no two computers can be simultaneously processing between their statements $L3$ and $L6$ for the same reason this is true in both Dijkstra's and Knuth's algorithms. Secondly, observe that the system cannot be blocked; for if none of the computers contending for access to its critical section has yet passed its statement $L3$, then after a point, the value of k will be constant, and the first contending computer in the cyclic ordering $(k, k + 1, \ldots, N, 1, \ldots, k - 1)$ will meet no resistance.

Finally, observe that no single computer can be blocked. Before any computer having executed its critical section can exit the area protected from simultaneous processing, it must designate as its unique successor the first contending computer in the cyclic ordering, assuring the choice of any individual contending computer within $N - 1$ turns.

Received February 1972; revised March 1972

References

1. Dijkstra, E.W. Solution of a problem in concurrent programming control. *Comm. ACM 8*, 9 (Sept. 1965), 569.
2. Knuth, D.E. Additional comments on a problem in concurrent programming control. *Comm. ACM 9*, 5 (May 1966). 321–322.
3. deBruijn, N.G. Additional comments on a problem in concurrent programming control. *Comm. ACM 10*, 3 (Mar. 1967), 137–138.