Concurrent control with "readers" and "writers"

Article *in* Communications of the ACM · October 1971

DOI: 10.1145/362759.362813 · Source: doi.acm.org

Pierre-Jacques Courtois
Université Catholique de Louvain - UCLouvain
91 PUBLICATIONS **2,125** CITATIONS
SEE PROFILE

David Parnas
Middle Road Software
302 PUBLICATIONS **19,328** CITATIONS
SEE PROFILE

Operating Systems
B. Randell Editor

Concurrent Control with "Readers" and "Writers"

P.J. Courtois,* F. Heymans, and D.L. Parnas*
MBLE Research Laboratory
Brussels, Belgium

The problem of the mutual exclusion of several independent processes from simultaneous access to a "critical section" is discussed for the case where there are two distinct classes of processes known as "readers" and "writers." The "readers" may share the section with each other, but the "writers" must have exclusive access. Two solutions are presented: one for the case where we wish minimum delay for the readers; the other for the case where we wish writing to take place as early as possible.

Key Words and Phrases: mutual exclusion, critical section, shared access to resources
CR Categories: 4.30, 4.32

* Present address: Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa 15213

Dijkstra [1], Knuth [2], and de Bruijn [3] have discussed the problem of guaranteeing exclusive access to a shared resource in a system of cooperating sequential processes. The problem they deal with has been shown to have a relatively simple solution using the "*P*" and "*V*" operations of Dijkstra [4]. We discuss two related problems of practical significance in which we recognize two classes of processes wishing to use the resource. The processes of the first class, named *writers*, must have exclusive access as in the original problem, but processes of the second class, the *readers*, may share the resource with an unlimited number of other readers.

Problem 1

We demand of our solution that no reader be kept waiting unless a writer has already obtained permission to use the resource; i.e. no reader should wait simply because a writer is waiting for other readers to finish. In this case the solution presented is quite simple, but our experience has shown that it is not easily arrived at. Numerous solutions, which have quite unreasonable complexity, have been proposed. The following solution resulted from several cycles among the authors in which each simplified a solution shown him by another. We present it in hope that others may be spared the effort of solving again this rather common problem. See Figure 1.

Please notice that *w* functions as a mutual exclusion semaphore for the writer but is only used by the first reader to enter the critical section and the last reader to leave it. It is ignored by readers who enter or leave while other readers are present. *mutex* ensures that only one reader will enter or leave at a time thereby eliminating the possibility of ambiguity about which process

Fig. 1

```
integer readcount ; (initial value = 0)
semaphore mutex, w ; (initial value for both = 1)

READER                                WRITER
P(mutex) ;
readcount := readcount + 1 ;
if readcount = 1 then P(w) ;
V(mutex) ;
                                      P(w) ;
        ...                                   ...
  reading is performed                writing is performed
        ...                                   ...
P(mutex) ;                            V(w) ;
readcount := readcount - 1 ;
if readcount = 0 then V(w) ;
V(mutex) ;
```

Fig. 2

```
integer readcount, writecount ; (initial value = 0)
semaphore mutex 1, mutex 2, mutex 3, w, r ; (initial value = 1)

READER                                  WRITER
P(mutex 3) ;                            P(mutex 2) ;
  P(r) ;                                writecount := writecount + 1 ;
    P(mutex 1) ;                        if writecount = 1 then P(r) ;
    readcount = readcount + 1 ;         V(mutex 2) ;
    if readcount = 1 then P(w) ;        P(w) ;
    V(mutex 1) ;
  V(r) ;
V(mutex 3) ;
        ...                                     ...
    reading is done                       writing is performed
        ...                                     ...
P(mutex 1) ;                            V(w) ;
readcount := readcount - 1 ;            P(mutex 2) ;
if readcount = 0 then V(w) ;            writecount := writecount - 1 ;
V(mutex 1) ;                            if writecount = 0 then V(r) ;
                                        V(mutex 2) ;
```

is responsible for adjusting *w*. *w* will be positive if and only if there are no readers and no writers present in the critical section.

Problem 2

Here we retain the requirement that writers must have exclusive access while readers may share, but we add the requirement that once a writer is ready to write, he performs his "write" as soon as possible. A solution to this problem cannot be a solution to Problem 1 because to meet this requirement a reader who arrives after a writer has announced that he is ready to write must wait *even if the writer is also waiting*. For the first problem it was possible that a writer could wait indefinitely while a stream of readers arrived. In this problem we give priority to writers and allow readers to wait indefinitely while a stream of writers is working. On general principles we require that the solution give priority to writers without making any assumptions about priority being built into the *V* routine. In other words, where several processes are waiting at a semaphore, we cannot predict which one will be allowed to proceed as the result of a *V* operation.

We propose the solution shown in Figure 2.

The reader should first note that the use of *mutex* 1 and *w* corresponds exactly to the use of *mutex* and *w* in the solution to Problem 1. The semaphore *r* is used to protect the act of entering the critical section in the same way that *w* is used to protect the shared resource in Problem 1. The first writer to pass *P*(*r*) will block readers from entering the section which manipulates *mutex* 1 and *w*. *mutex* 2 is used here for writers just as *mutex* is used for readers in Problem 1. *mutex* 3 is necessary because of our absolute insistence on priority for writers. Without *mutex* 3 we have the possibility that a writer and one or more readers will be simultaneously waiting for a *V*(*r*) to be done by a reader. In that event we could not guarantee priority to the writer. *mutex* 3 guarantees a reader exclusive access to the block of code from "*P*(*r*)" to "*V*(*r*)" inclusive. As a result there will be at most one process ever waiting at *r*, and the result of a *V* is clear.

Final Remarks

The reader will note that the above solutions do not guarantee a FIFO discipline for the writers. To provide such a guarantee we must either assume further properties of the *V* operation or make use of an array of *n* semaphores where *n* is the number of writers.

Acknowledgment. We are grateful to A.N. Habermann of Carnegie-Mellon University for having shown us an error in an earlier version of this report.

References

1. Dijkstra, E.W. Solution of a problem in concurrent programming control. *Comm. ACM 8*, 9 (Sept. 1965), 569.
2. Knuth, D.W. Additional comments on a problem in concurrent programming control. *Comm. ACM 9*, 5 (May 1966), 321–322.
3. de Bruijn, N.G. Additional comments on a problem in concurrent programming control. *Comm. ACM 10*, 3 (Mar. 1967), 137–138.
4. Dijkstra, E.W. The structure of the "THE"-multiprogramming system. *Comm. ACM 11*, 5 (May 1968), 341–346.