Mutual Exclusion with Linear Waiting Using Binary Shared Variables

James E. Burns

School of Information and Computer Science
GEORGIA INSTITUTE OF TECHNOLOGY
Atlanta, Georgia 30332

Abstract: The problem of mutual exclusion among N asynchronous, parallel processes using only shared binary variables for communication is considered. Upper and lower bounds of N+1 and N shared binary variables, respectively, are shown for the problem of mutual exclusion with linear waiting. Lockout-free mutual exclusion is shown to require at least N shared binary variables when the primitive operations are suitably restricted. This latter bound is tight for N=2.

Introduction

The problem of mutual exclusion among asynchronous, parallel processes has been studied for some time, [1,2,3]. Recent, work, [4,5], has explored the bounds on the shared space required to implement various mutual exclusion problems. These papers use a generalized test-and-set operation, which allows arbitrary tranformations of the values in the shared space by a single indivisible operation. This report examines the space bounds for the more realistic case, (given today's technology), in which indivisible operations are restricted to single binary variables.

An algorithm is presented which solves the problem of mutual exclusion with linear waiting using N+1 shared binary variables, only one of which requires a special primitive, test-and-set-high. The test-and-set-high primitive is of particular interest since it is available on several existing machines. The algorithm is shown to be near optimal in shared space requirements - at least N shared binary variables are needed. The less constrained problem of lockout-free mutual exclusion is also shown to require N shared binary variables when the primitive operations are restricted to test-and-set-high and test-and-set-low. This latter bound is shown to be tight for N=2.

A formalism for defining the problems studied here will not be given in this short report, but the reader is directed to [5]. Related formalisms appear in [4,6]. In this report, an N-system is a set of N processes which proceed asynchronously and communicate only by means of shared variables. Each process has a set of states which it may assume. This set is divisible into mutually disjoint subsets called the critical region, the remainder region and the trying region. (Note: the exit region concept of [5] has been omitted for simplicity. We allow the critical region to be extended to encompass the exit region at the expense of slightly weakening the lower bound results.) A process cylces through

the remainder, trying, critical and back to the remainder region, possibly bypassing the trying region. A process may halt only in the remainder region. An N-system may satisfy one or more of the following properties:

P1 Mutual Exclusion: No two processes may be in their critical regions at the same time.

P2 Deadlock-free: If any process is in its trying region, then at some future time some process must enter its critical region.

P3 Lockout-free: If a process is in its trying region, then at some future time it must enter its critical region.

P4 Bounded Waiting: There is a constant k such that if a process is in its trying region, then that process will enter its critical region before any other process has entered its critical region more than k times. When k=1, this property is call Linear Waiting.

A binary N-system is an N-system in which all of the shared variables are binary and in which the only allowed indivisible operations reference at most one shared variable. The eight allowed operations on binary variables are listed below:

Function	Without Return		With Return	
$(0,1) \rightarrow (0,1)$	NOP	(no operation)	RD	(read)
$(0,1) \rightarrow (0,0)$	WRL	(write low)	TSL	(test-and-set-low)
$(0,1) \rightarrow (1,1)$	WRH	(write high)	TSH	(test-and-set-high)
$(0,1) \rightarrow (1,0)$	XCH	(exchange)	TXC	(test-and-exchange)

We may think of these primitives as functions which have the side effect of modifying their argument as specified. The functions may either return no value or may return the value of the calling argument. Most machines allow only the NOP, WRL, WRH and RD operations as indivisible operations. Variables which are used in this way will be call read/write variables. If, in addition, the TSH operation is used on a variable, the variable will be called a test-and-set-high variable. If both TSH and TSL are used, the variable will be called a test-and-set-high/low variable.

An Upper Bound

Theorem 1: There is a binary N-system with N+1 shared variable which satisfies the properties of mutual exclusion and linear waiting. Furthermore, this may be done with N read/write variables and one test-and-set-high variable.

Proof sketch: It should be clear that the algorithm presented on the next page defines a binary N-system with the number and type of shared variables as stated in the theorem. Observe that a process may enter its critical region only upon a "successful", (i.e, false return), test-and-set-high on KEY or upon finding its own TRY variable reset to false.

Since only one of these conditions can occur when a process leaves its critical region, and cannot occur otherwise, mutual exclusion is guaranteed. This argument also implies that the system is deadlock-free, for a process leaving its critical region either sets KEY to false, allowing any trying process to enter, or selects a new process to enter by setting the TRY variable of that process to false. (The selected process must enter since its TRY variable cannot be reset to true.)

As each process leaves its critical region it scans the TRY variables to find the next higher numbered process, (in cyclic order), which is in its trying region. It is possible for a newly trying process to be skipped in this scan, but only by a process that was already in its critical region when the new process set its TRY variable to true. A trying process cannot be skipped by any process which enters its critical section at a later point. The cyclic order of selecting processes from those trying thus gives linear waiting. □

Mutual Exclusion with Linear Waiting

```
Shared Variables:  Boolean KEY;                    # initially false
                   Boolean array TRY[0:N-1];       # initially false

Algorithm for process i:
  begin Boolean T;  integer j;
     while true do begin
        REMAINDER_REGION;
        TRY[i] := true;
        T      := true;
        while TRY[i] and T do
           T := TSH[KEY];
        CRITICAL_REGION;
        TRY[i] := false;
        j := i+1 mod N;
        while not TRY[j] and j≠i do
           j := j+1 mod N;
        if j=i then KEY    := false
               else TRY[j] := false
     end
  end
```

Lower Bounds

Theorem 2: A binary N-system with the properties of mutual exclusion and bounded waiting must have at least N shared variables, $N \geq 2$.

Proof sketch: (Note: lower bound proofs really require a formal system. Here we will simply give a very intuitive proof sketch. A more rigorous proof based on the formalism of [5] will appear in a later technical report.) Suppose the theorem is false. Consider a computation sequence in which all N processes proceed to a point just before modifying a

shared variable. (If some process enters its critical region without modifying a shared variable then mutual exclusion can be violated). Since there are less than N shared variables, at least two processes must modify the same variable. Extend the computation by one step for each of two such processes. Their joint action either results in no change in the shared variable or changes it to its alternate value. In the first case, the chosen pair of processes is "hidden" from the other processes. A finite extension to the computation sequence can then be found which causes some other process to enter its critical region a number of times greater than any given bound k. (Note: this argument does not hold for N=2. For this case we rely on a theorem by Cremers and Hibbard which implies that lockout-free mutual exclusion requires three states in the shared variables. Since a lower bound on the lockout-free property also holds for bounded waiting, the theorem is true for N=2.) In the second case, one of the processes must not have changed the value of the shared variable at all. It is then hidden from the second process and bounded waiting can again be shown to be violated. The supposition is thus false and the theorem must be true. □

Corollary: A binary N-system with the properties of mutual exclusion a linear waiting must have at least N shared variables.

Proof: Direct from Theorem 2 and the definitions of bounded and linear waiting.

Theorem 3: A binary N-system in which all shared variables are test-and-set-high/low and which has the properties of mutual exclusion and lockout-free must have at least N shared variables, $N \geq 2$.

Proof sketch: By induction on N. From a theorem by Cremers and Hibbard [4], lockout-free mutual exclusion for two processes requires at least three distinct states in the shared variables. Therefore, at least two binary variables are necessary, and the theorem is true for N=2. Suppose the theorem true for N=k. Suppose a binary k+1-system with only k shared variables satisfies lockout-free mutual exclusion. There must be a computation sequence using only k of the processes which causes each of the shared variables to change infinitely often, (otherwise, the inductive hypothesis would be violated). Consider the k+1st process. It must modify one of the shared variables before entering its critical region, (see the proof of Theorem 2). But by interleaving the steps of the k+1st process with the infinite sequence previously defined a new sequence can be found with the property that just before the k+1st process writes a variable, that variable takes on the value about to be written. The k+1st process can thus never be "seen" by the other processes and will be locked out forever. The supposition is therefore false, and the induction holds. □

Lockout-free mutual exclusion has an upper bound of N+1 shared binary variables from Theorem 1 and the definitions. However, for N=2, this bound can be improved to N, as shown by the following algorithm.

Lockout-free Mutual Exclusion for Two Processes

```
Shared Variables:  Boolean V1, V2;      #initially false

Algorithm for each process:
  begin Boolean X,Y;
     while true do begin
        REMAINDER REGION;
        X := TSH(V1);                   # r
        if X then begin
           repeat
              Y := TSH(V2);             # s
              if Y then X := TSH(V1)    # t
           until not X or not Y;
           if not Y then begin
              repeat
                 X := TSH(V1)           # u
              until not X;
              V2 := false               # v
           end
        end;
        CRITICAL REGION;
        V1 := false;                    # c
        repeat
           Y := V2                      # d
        until not Y
     end
  end
```

As with many parallel algorithms, this one is somewhat tedious to prove correct. A bare outline of one method of proof will be given here.

A state of the system can be specified by a four-tuple, (a,b,x,y), where a and b are the states of the two processes and x and y are the values of V1 and V2, respectively. The state of a process is well specified by the next operation to be made on a shared variable, as denoted by the comment letters in the algorithm. The initial system state is then (r,r,0,0). The complete reachability graph, containing 34 distinct nodes, can easily be constructed. The arcs of the graph may be labelled with the name of the process causing the particular transition. Mutual exclusion can be verified immediately from the graph by observing that there are no nodes of the form (c,c,x,y). The lockout-free property is equivalent to satisfying the following conditions on the graph: (1) Every cycle whose nodes are entirely of the form (a,r,x,y) must contain a node of the form (c,r,x,y). (2) Every cycle of the form (r,b,x,y) must contain a node like (r,c,x,y). (3) Every cycle containing arcs labelled with the names of both processes must contain a node of the form (c,b,x,y) and a node of the form (a,c,x,y). These conditions can be shown to hold for the graph in question, so the algorithm does solve lockout-free mutual exclusion for two processes.

Conclusions

Tight upper and lower bounds on shared space for binary N-systems have been shown for the problems of mutual exclusion with linear waiting and, (with restricted primitives), lockout-free mutual exclusion. Although there is some room for improvement for both problems, it may be difficult to remove the gap of a single binary variable.

An especially interesting, and apparently difficult, problem is to find a lower bound on lockout-free mutual exclusion without restricting the allowed primitives. The best bound currently known is $\log_2(N/2)$, which comes directly from [5]. The upper bound of N+1 shared variables may also be subject to improvement.

Acknowledgment: Paul Jackson originally suggested that the test-and-set-high primitive deserved to be examined because of its availability on existing machines. Paul independently derived an algorithm for mutual exclusion with linear waiting which uses $N + 1 + \log_2 N$ shared binary variables.

References

[1] Dijkstra, E., "Solution of a Problem in Concurrent Control", CACM 8 (Sep 1965), P. 569.

[2] Knuth, D., "Additional Comments on a Problem in Concurrent Control", CACM 9 (May 1966), p.321-322.

[3] Eisenberg, M. and M. McGuire, "Further Comments on Dijkstra's Concurrent Programming Control Problem", CACM 15 (Nov 1972), p. 999.

[4] Cremers, A. and T. Hibbard, "An Algebraic Approach to Concurrent Programming Control and Related Complexity Problems", University of Southern California Technical Report (Nov 1975).

[5] Burns, J., M. Fischer, P. Jackson, N. Lynch and G. Peterson, "Shared Data Requirements for Implementation of Mutual Exclusion Using a Test-and-set Primitive", Proc. of the 1978 International Conf. on Parallel Processing (to appear).

[6] Miller, R. and C. Yap, "Formal Specification and Analysis of Loosely Connnected Processes", IBM Research Report RC 6717 (Sep 1977).