MUTUAL EXCLUSION USING INDIVISIBLE READS AND WRITES*

JAMES S. BURNS  
Computer Science Department  
Indiana University  
Bloomington, Indiana 47405

NANCY A. LYNCH  
Information & Computer Science  
Georgia Inst. of Technology  
Atlanta, Georgia 30332

ABSTRACT

The shared space (the number of distinct values taken on by the set of shared variables) requirements of Dijkstra's original mutual exclusion problem are examined. It is shown that 2N shared states are necessary and sufficient to solve the problem of deadlock-free mutual exclusion for N processes using only individual reads and writes of shared variables for communication.

INTRODUCTION

The first solution to the mutual exclusion problem was given in 1965 by Dijkstra [1]. The original definition of the problem requires that N processes be synchronized so that no two processes are simultaneously executing portions of their code which are called "critical sections". Processes execute asynchronously; that is, they execute at independent, finite, non-zero rates, possibly varying over time. A process may halt if it is not executing its critical section or the part of its code which is devoted to synchronization. To prohibit trivial solutions, the system of processes is required to be deadlock-free.

In Dijkstra's paper, and subsequent papers by Knuth [2], de Bruijn [3] and Eisenberg and McGuire [4], all communication between processes was required to be through shared variables. The only actions allowed on shared variables were reads and writes, which were assumed to be indivisible.

More recent work has examined the mutual exclusion problem using more powerful primitive actions for accessing shared variables. One of the primary concerns in the papers of Cremers and Hibbard [5], Burns, Fischer, Lynch, Jackson and Peterson [6], Burns [7], Peterson [8], and others has been the number of shared states required to solve certain variations of the mutual exclusion problem. (The number of shared states is the number of distinct values taken on jointly by the set of shared variables.) In this

*This research was supported in part by Army Research Office Contract DAAG29-79-C-0155 and NSF Grants MCS77-15628 and MCS78-01699.
paper we examine the number of shared states required for Dijkstra's original problem.

It is interesting to note that although the algorithms in papers [1-4] solve slightly different problems, they all use exactly the same number of shared states, \(N^3N\) (one \(N\)-valued variable and \(N\) three-valued variables). (Note: Dijkstra [1] uses \(N\) pairs of binary variables, but each pair takes on only three possible values.) We will show that \(N\) binary variables are necessary and sufficient to solve the problem of deadlock-free mutual exclusion.

**READ/WRITE-SYSTEMS**

In order to define precisely what we mean by asynchronous systems communicating by shared variables, we will briefly introduce a formal model based on the models presented in [6] and [9]. We have omitted definitions which are not important for the present paper. In the sequel, let \([n] = \{1, 2, ..., n\}\), for any positive integer \(n\).

A system is a 4-tuple, \(S = (P, V, q_0, \phi)\), where \(P\) is a finite set of processes, \(V\) is a finite set of variables, \(q_0\) is the initial instantaneous description of \(S\) and \(\phi\) is the transition function of \(S\). We will always let \(N = |P|\) and \(M = |V|\) and index the processes of \(S\) by \([N]\) and the variables by \([M]\). The possibly infinite set, \(X_i\), of states of \(P_i\) is partitioned into mutually disjoint sets \(R_i, T_i, C_i\) and \(E_i\). \(R_i, T_i, C_i\) and \(E_i\) are called the remainder region, trying region, critical region and exit region, respectively. The set of values of the \(j\)th variable of \(V\) is \(V_j\). An instantaneous description (id) of \(S\) is an \((N+M)\)-tuple, \(q = (x_1, x_2, ... , x_N, v_1, v_2, ... , v_M)\), where \(x_i \in X_i\) for each \(i \in [N]\) and \(v_j \in V_j\) for each \(j \in [M]\); we use \(X_i, V_j,\) and \(V\) (ambiguously) as projection operators defined by \(X_i(q) = x_i, V_j(q) = v_j\) and \(V(q) = (v_1, ... , v_M)\).

Let \(Q\) be the set of all ids of \(S\). Then \(\phi\) is a total function, \(\phi: P \times Q \rightarrow Q\), satisfying the following conditions. Let \(i\) and \(j\) be integers such that \(i \neq j\) and \(i, j \in [N]\), and let \(q\) be an id of \(S\). If \(q' = \phi(i, q)\) then we write \(q \Rightarrow q'\) and require that

1. \(X_j(q') = X_j(q)\)
(2) If \( X_i(q) \in R_i \cup T_i \), then \( X_i(q') \in T_i \cup C_i \).
(3) If \( X_i(q) \in C_i \cup E_i \), then \( X_i(q') \in E_i \cup R_i \).

Condition (1) requires that processes communicate only by the shared variables. Conditions (2) and (3) allow processes to loop only within the trying and exit regions, which are used to execute synchronization protocols for the critical region. The internal structure of the remainder and critical regions are not important here and are suppressed.

A system \( S=(P,V,q_0,\delta) \) is a read/write-system if for every \( i \in [N] \), \( X_i \) can be partitioned into two disjoint sets Read\(_i\) and Write\(_i\) such that the following conditions hold for every \( i \) of \( S \). If \( X_i(q) \in \text{Read}_i \), then there exists a \( k \in [M] \) such that

(4) \( V(\delta(i,q)) = V(q) \),
(5) for every \( j \) of \( S \) with \( X_j(q') = X_i(q) \) and \( V_k(q') = V_k(q) \),
\( X_j(\delta(i,q')) = X_i(\delta(i,q)); \)

we say that the transition \( q \rightarrow \delta(i,q) \) is a read of the \( k \)th variable by \( P_i \). If \( X_i(q) \in \text{Write}_i \), then there exists \( k \in [M] \) such that

(6) for every \( j \in [N] \), if \( j \neq k \) then \( V_j(\delta(i,q)) = V_j(q) \).
(7) for every \( j \) of \( S \) with \( X_j(q') = X_i(q) \),
\( V_k(\delta(i,q')) = V_k(\delta(i,q)); \)
(8) for every \( j \) of \( S \) with \( X_j(q') = X_i(q) \),
\( X_j(\delta(i,q')) = X_i(\delta(i,q)); \)

we say that the transition from \( q \) to \( \delta(i,q) \) is a write of the \( k \)th variable by process \( i \) and that \( P_i \) is about to write the \( k \)th variable at \( q \).

In order to reason about the behavior of a system, we wish to specify individual computations, removing the nondeterminism of the transition function of \( S \). This is done with schedules. A schedule of \( S \) is a finite or infinite sequence of process indices (elements of \([N]\)). If \( q_1 \) is an id of \( S \) and \( h=i_1i_2... (h=i_1i_2...i_k) \) is an infinite (finite) schedule of \( S \), then \( q_1q_2... (q_1...q_{k+1}) \) is the computation sequence from \( q_1 \) by \( h \), where \( q_{j+1} = \delta(i_j,q_j) \) for \( j = 1,2,... (j=1,...,k) \). If \( h \) is finite, then \( r(q,h) \) is the final id in the computation sequence from \( q \) by \( h \). We say that id \( q' \) is reachable from id \( q \) if there exists a finite schedule \( h \) such that \( q' = r(q,h) \). A schedule, \( h \), of \( S \) is
admissible from id q if for every finite prefix, \( h_1 \), of h (h = \( h_1 h_2 \)), and for every \( i \in [N] \), \( X_i(r(q,h_1)) \notin R_i \) implies that i occurs in \( h_2 \). Admissibility requires that no process halts outside its remainder region.

**DEADLOCK-FREE MUTUAL EXCLUSION**

A system S satisfies mutual exclusion if for every id q of S reachable from \( q_0 \) and every pair \( i,j \in [N] \), \( X_i(r(q,h)) \notin C_j \) and \( X_j(r(q,h)) \notin C_i \) implies that \( i=j \).

Process i is said to change regions from q by h if there exists finite prefixes \( h_1 \) and \( h_2 \) of h such that \( X_i(r(q,h_1)) \) is in a different region than \( X_i(r(q,h_2)) \). A system S is deadlock-free if for every id q, reachable from \( q_0 \) and every non-null schedule, h, admissible from q, some process changes regions from q by h.

**THE UPPER BOUND**

The algorithm below is in a Pascal-like notation. The algorithm begins execution with all shared variables set to "down" and with process i executing the following program for each \( i \in [N] \).

```pascal
program Process_i;
  type flag = (down,up);
  shared var F : array [1..N] of flag;
  var j : 1..N;
  begin
    while true do begin
      1: F[i] := down;
      2: remainder; (* remainder region *)
      3: F[i] := down;
      4: for j := 1 to i-1 do
          if F[j] = up then goto 3;
      5: F[i] := up;
      6: for j := 1 to i-1 do
          if F[j] = up then goto 3;
      7: for j := i+1 to N do
          if F[j] = up then goto 7;
      8: critical; (* critical region *)
      end.
  end.
```

**Theorem 1:** For every integer \( N \geq 0 \) there exists a read/write-system of \( N \) processes and \( N \) binary shared variables which solves the problem of deadlock-free mutual exclusion.
Proof: For any integer \( N > 0 \), it is clear that the above algorithm defines a read/write-system with \( N \) processes and \( N \) binary shared variables. We must show that the system is deadlock-free and satisfies mutual exclusion.

Suppose deadlock can occur. Then we can reach an id, \( q \), at which at least one process is not in remainder and a schedule \( h \), admissible from \( q \), such that no process changes region from \( q \) by \( h \). Since the only backward branches in each process's program occur in the trying region, we observe that for each \( i \in [N] \), either \( X_i(q) \in R_i \) and \( i \) does not occur in \( h \) or \( X_i(q) \in T_i \) and \( i \) occurs infinitely often in \( h \). We call the set of processes which are not in remainder at \( q \) "active".

For each \( i \in [N] \), define the following subsets of \( T_i \). \( A_i = \) the sets of states of \( P_i \) corresponding to the statements labeled 3 and 4. \( B_i = \) the sets of states of \( P_i \) corresponding to the statement labeled 7. We note that if \( P_i \) reaches \( B_i \), then it will remain there for the rest of the computation and \( P[i] \) will be continuously equal to "up". Let \( m = \min \{ i \in [N] : P_i \) is active at \( q \} \). Since \( m \) will eventually detect that no \( P[i] = \) up for \( i \in [m-1] \), \( P_m \) will reach \( B_m \) after a finite prefix, \( h_1 \), of \( h \) (\( h = h_1h_2 \)). (That is, \( X_m(q') \in B_m \), where \( q' = r(q,h_1) \). After some finite prefix, \( h_2 \), of \( h_2 = h_3h_4 \), every active \( P_i \) will either be in \( B_i \) or will begin cycling forever in \( A_i \) with \( P[i] = \) down, since all higher indexed processes which do not reach \( B_i \) will detect \( P[m] = \) up. Let \( n = \max \{ i \in [N] : X_i(q') \in B_i \} \), where \( q'' = r(q',h_3) \). Now \( P_n \) will find all \( P[i] = \) down for \( i \in [n+1,\ldots,N] \), so \( P_n \) will change regions from \( q'' \) by \( h_4 \), contradicting our supposition. Therefore, deadlock cannot occur.

Suppose that mutual exclusion may be violated. Then there must be values \( i,j \in [N] \) such that \( i \neq j \) and a finite schedule \( h \) such that \( q = r(q_0,h) \) and \( X_i(q) \in C_i \) and \( X_j(q) \in C_j \). Let \( D_i = \) the set of states of \( P_i \) corresponding to statements 6, 7 and 8 of the algorithm, and \( D_j \) be similarly defined for \( P_j \). \( P_i \) may enter and leave \( D_i \) several times before reaching its critical region, but there must be an id at which \( P_i \) enters \( D_i \) for the last time before going critical. Let \( q_a \) be this id for \( P_i \), and let \( q_b \) be a similar id for \( P_j \), in the computation sequence.
$q_0q_1...q_k$ from $q_0$ by $h$ ($q=q_k$). We may assume without loss of generality that $a \leq b$. But then for every $c, a \leq c \leq b$, $P(i)=u$ at $q_c$. Since $P_j$ must test $P[i]$ after entering $D_j$ (either at statement 6 of 7), $P_j$ cannot go critical in the computation sequence $q_0q_{b+1}...q_k$, contradicting our supposition. Therefore the algorithm also satisfies mutual exclusion and the theorem is proved. \[\square\]

**The Lower Bound**

Some additional definitions will be needed in the lemmas leading up to the lower bound theorem. As before, we let $S$ be a read/write-system. Let $q_1$ be an id of $S$, $h$ be a schedule of $S$, $i \in [N]$, $v \in V$ and $q_1q_2...$ be the computation sequence from $q_1$ by $h$. If there exist positive integers $j \leq k$ such that $q_j\rightarrow q_{j+1}$ is a write of $v$ by $P_j$ and $q_k\rightarrow q_{k+1}$ is a write of $v$, and if for all $n$, $j \leq k$, $q_n\rightarrow q_{n+1}$ is not a read of $v$ by any process other than $P_i$, then we say that the write of $v$ by $P_i$ at $q_j$ is obliterated from $q$ by $h$.

Let $q$ be an id of $S$, $h$ be a schedule of $S$, and $i \in N$. If $X_i(q) \in R_i$ and every write by $P_i$ is obliterated from $q$ by $h$, then $P_i$ is invisible from $q$ by $h$. If $i \in [N]$, $q$ is an id of $S$, $h_1, h_2$ are schedules of $S$ ($h_1$ finite) such that $P_i$ is invisible from $r(q, h_1)$ by $h_2$, then $P_i$ is hidden from $q$ by $h_1h_2$.

**Lemma 1**: Let $S$ be a read/write-system, $h$ be a finite schedule of $S$ and $P_i$ be a process of $S$ which is hidden from $q_0$ by $h$, and let $q=r(q_0, h)$. Then there is an id $q'$ of $S$ reachable from $q_0$ such that $X_i(q') \in R_i$, $V(q')=V(q)$ and $X_j(q')=X_j(q)$ for all $j \neq i$.

**Proof**: Let $h_1$ be the longest prefix of $h$ for which $X_i(r(q_0, h_1)) \in R_i$ ($h_1$ exists since $P_i$ is hidden from $q_0$ by $h$) so that $h = h_1h_2$. Let $h_3$ be the schedule equivalent to $h_2$ with all occurrences of $i$ removed, and let $h' = h_1h_3$. Now $q'=r(q_0, h')$ meets the requirements of the lemma since $P_i$ cannot have left the remainder region since $r(q_0, h_1)$. Note that $V(q')=V(q)$ because all writes by $P_i$ in the computation from $r(q_0, h_1)$ to $q$ are obliterated. Also, $X_j(q')=X_j(q)$ for $j \neq i$.
because no process can have read anything written by \( P_1 \) since 
\( r(q_0, h_1) \).

Let \( S \) be a read/write-system, \( q \) be an id of \( S \) and \( i \in [N] \). If \( P_1 \) is about to write \( v \in [N] \) at \( q \), then we say that \( v \) is covered at \( q \) by \( P_1 \).

**Lemma 2:** Let \( S \) be a read/write-system with at least two processes which solves deadlock-free mutual exclusion, \( h \) be a finite schedule of \( S \) and \( P_1 \) be a process of \( S \) hidden from \( q_0 \) by \( h \). If \( P_1 \) goes critical on its own from \( q = r(q_0, h) \) by a schedule \( h_1 = i^K \), then \( P_1 \) must write some variable in the computation sequence from \( q \) by \( h_1 \) which is not covered by any other process at \( q \).

**Proof:** Suppose \( P_1 \) goes critical from \( q \) by schedule \( h_1 \) without writing any variable which is not covered by some other process at \( q \). Let \( h_2 \) a schedule consisting of exactly one step of each process other than \( P_1 \). Then every write of \( P_1 \) from \( q \) is obliterated from \( q \) by \( h_1 h_2 \), so \( P_1 \) is hidden from \( q \) by \( h_1 h_2 \). By Lemma 1, there is a reachable id \( q' \) which looks like \( q' = r(q, h_1 h_2) \) to all the other processes but has \( P_1 \) in remainder. By no deadlock, some other process \( P_j \neq P_1 \) can go critical from \( q' \) by schedule \( h' \). But then \( r(q', h') \) has both \( P_1 \) and \( P_j \) critical, contradicting mutual exclusion.

Let \( S \) be a read/write-system, \( q \) be an id of \( S \), \( h \) be a finite schedule of \( S \) and \( W \) be a subset of \( V \). We say that \( W \) is nullified from \( q \) by \( h \) if for every \( w \in W \) there is a process which is hidden from \( q \) by \( h \) and which covers \( w \) at \( r(q, h) \).

**Lemma 3:** Let \( S \) be a read/write-system with \( N \geq 2 \) processes which solves deadlock-free mutual exclusion, and let \( q \) be any reachable id of \( S \) at which all processes of \( S \) are in their remainder regions. For every \( K, 1 \leq K \leq N \), there is a finite schedule \( h \) of \( S \) using only processes \( P_1, P_2, \ldots, P_K \) such that \( K \) variables are nullified from \( q \) by \( h \).

**Proof:** The proof is by induction on \( K \), the number of variables nullified.
BASIS. Let \( K=1 \). By no deadlock, there must be a finite schedule \( h' \) consisting only of 1's such that \( P_1 \) goes critical at \( r(q_i, h') \). By Lemma 2, there must be a prefix, \( h'' \) of \( h' \) such that \( P_1 \) is hidden from \( q \) by \( h'' \) and covers some variable, \( w \), at \( r(q_i, h'') \). But then \( \{w\} \) is nullified from \( q \) by \( h'' \) and the lemma holds for \( K=1 \).

INDUCTIVE STEP. Assume Lemma 3 holds for \( K=k-1 \). By the inductive assumption, there is a finite schedule \( h_0 \) from \( q \) using only processes \( P_1, \ldots, P_{k-1} \) such that a set, \( W_1 \), of \( k-1 \) variables is nullified from \( q_0 \) by \( h_0 \). Let \( q_1 = r(q_0, h_0) \).

From \( q_1 \) we can successively reach id's \( q_2, q_3, \ldots \) by finite schedules \( h_1, h_2, \ldots \) such that \( q_{i+1} = r(q_i, h_i) \), where \( h_i \) is defined in the following way. For each \( i=1, 2, \ldots \), let \( h_i \) begin with the prefix \( 123 \ldots (k-1) \). From \( r(q_i, 123 \ldots (k-1)) \), find an extension of \( h_i \) which returns \( P_1, \ldots, P_{k-1} \) to their remainder regions (by no deadlock this extension exists). Finally, complete \( h_i \) by appending a schedule which nullifies a set, \( W_{i+1} \), of \( k-1 \) variables at \( q_{i+1} \). The final portion of \( h_i \) exists by the inductive assumption.

For each \( i=1, 2, \ldots \), Lemma 1 and Lemma 2 imply that \( P_k \) can be moved on its own by a schedule \( s_i \) from each \( q_i \) such that \( P_k \) is ready to write some variable, \( w_i \), which is not in \( W_i \). We claim there are integers \( i \) and \( j \), where \( 0<i<j \), such that \( w_i \not\in W_j \).

Suppose not. Then \( \{w_1, \ldots, w_{i-1}\} \) is a subset of \( W_i \), for \( i=1, 2, \ldots \). This implies that \( w_1, \ldots, w_i \) are distinct since \( w_i \) is not in \( W_i \). But then \( W_{i+2} \) would contain a subset of \( N+1 \) variables, which is impossible (one process is required to nullify each variable) so the claim holds.

Consider the computation from \( q_i \) produced by schedule \( s = s_1 h_1 h_{i+1} \ldots h_j \), where \( i \) and \( j \) are chosen as in the claim. The first part of \( h_i \) obliterates all writes of \( P_k \) in the computation from \( q_i \) by \( s_1 \), so \( P_k \) is hidden from \( q_i \) by \( s \). Now variables in \( W_j \cup \{w_i\} \) are nullified from \( q_i \) by \( h_j \) and \( |W_j \cup \{w_i\}| = k \), by the claim. Thus we have \( k \) variable nullified from \( q \) by \( h_0 h_1 \cdots h_{i-1} s_i h_i h_{i+1} \cdots h_j \), and the lemma is proved. \( \square \)

**Theorem 2**: If \( S \) is a read/write-system with at least two processes and \( S \) solves deadlock-free mutual exclusion, then \( S \) must have at
least as many variables as processes.

Proof: By no deadlock, there is an id q reachable from q0 such that all processes of S are in their remainder regions at q. We may then apply Lemma 3 for q to find another reachable id which covers N distinct variables. □

FURTHER REMARKS

The techniques used in the proof of Lemma 3 may be applied to prove similar theorems for other exclusion problems. The following characteristics appear to be needed. (1) A basis for the induction. It must be possible to show that some number of variables can be covered by a set of hidden processes. (2) For the inductive step, a lemma similar to Lemma 2 must be proved.

Secondly, we must have that a subset of N-1 processes of a system of N processes forms a system of the type required. This second condition does not hold for the dining philosophers problem, for example.

Although the algorithm presented in this paper does use fewer shared states than previous solutions, it does not meet one requirement that was imposed by Dijkstra. In [1], Dijkstra states that "The solution must be symmetrical between the N computers; as a result we are not allowed to introduce a static priority". The algorithm given here does not meet this condition because process 1 apparently has the highest priority (it will never be locked out). A future paper will explore the relationships between symmetry and solutions to synchronization problems. One difficulty is finding a useful formal definition of symmetry which coincides with our intuitive understanding of the concept.

Acknowledgement:

We would like to thank Mike Fischer for helpful comment and criticism.
REFERENCES


