Fundamenta Informaticae 40 (1999) 407–422
IOS Press

Generalized Mutual Exclusion with Semaphores Only

E.T. Ordman[†], **E. Eberbach**[*], **A. Anwar**[†]
Department of Mathematical Sciences
The University of Memphis
Memphis, TN 38152, U.S.A.

Abstract. The paper deals with a generic solution of the Mutual Exclusion Problem using semaphores only. We use in the solution weakly fair binary semaphores and (not necessarily fair) semaphores with initial value k. All semaphores are *simple* in that the **P** and **V** operations on them are paired in the natural way avoiding split semaphores. No auxiliary shared variables are used. We define a general form of the mutual exclusion problem. We claim: (1) All mutual exclusion problems can be solved using only simple semaphores. (2) All mutual exclusion problems can be solved fairly (with bounded waiting) using simple semaphores (weakly fair binary and initially–k semaphores).

We give some bounds on how many semaphores are needed for standard problems. The solutions given may be inefficient: a mutual exclusion problem which can be solved in linear time and space with shared variables may require exponentially many semaphores.

Keywords: mutual exclusion, semaphore, weakly fair semaphore, bounded waiting.

1. Introduction

This paper concerns mutual exclusion of concurrent processes; see e.g. [1]. Some authors have studied enforcing mutual exclusion using shared variables (e.g. [8]) or using weak semaphores together with shared variables ([9], [10], [15]). We propose to enforce mutual exclusion with bounded waiting using only semaphores, some of which are guaranteed to be weakly fair, and

[*]work done while on leave from the Jodrey School of Computer Science, Acadia University, Wolfville, NS, Canada B0P 1X0, eugene.eberbach@acadiau.ca, research supported in part by NSERC under grant OGP0046501
[†]Address for correspondence: Department of Mathematical Sciences, The University of Memphis, Memphis, TN 38152, U.S.A.

to restrict use of these semaphores to very simple ways (e.g., avoid such constructs as 'split semaphores').

Of course, there are other mechanisms for synchronization between parallel statements (see, e.g. [1], [3], [5], [7], [16]), including conditional critical regions, monitors, channels, rendezvous, path expressions, COSY, Petri nets, Dijkstra's guarded commands, process algebras, temporal and linear logics. They can be more structured, powerful, and leading to cleaner and more readable programs than semaphores, but they are not a subject of this paper. Semaphores belong to the oldest and the best understood mechanisms, thus they can be used as a theoretical yardstick to compare other methods. However, besides readability and difficulties to develop well-behaving programs with semaphores, their use has other shortcomings. As is shown in the paper, the attempt to generate the automatic solution of a general mutual exclusion problem using semaphores only, may lead to exponential time and space complexities.

Some of our definitions will be unusually restrictive so as to delineate the subject of this paper without having to use modifiers too repetitively.

*Definitions.***Mutual exclusion.** We will consider finite sets of computer processes, each of which has a distinct name. Each process will be a simple infinite loop of the form

```
while true do
   begin
       entry_section;
       critical_section;
       exit_section;
       remainder_section;
   end;
```

By a *mutual exclusion problem* $\mathcal{P} = (\mathrm{PR}, S)$ we mean a set of processes $\mathrm{PR} = \{\mathrm{PR}[i] \mid i = 1, 2, \ldots, n\}$ together with a collection S of nonempty sets $\{S_j \mid S_j \subseteq \mathrm{PR}, j = 1, 2, \ldots, m\}$ which cannot be in their critical sections at the same time and no $S_j \subseteq S_k$ for $j \neq k$. The sets S_j will be called *minimally mutually excluding subsets* of $\mathcal{P}$, and an S_j may be called an MMES for brevity. We relate this to the usual understanding of a mutual exclusion problem as follows:

A set of processes $\mathrm{C} = \{\mathrm{PR}[\mathrm{i_k}] \mid \mathrm{k} = 1, 2, \ldots, \mathrm{r}\}$ *may be in their critical sections at the same time if and only if the set* C *does not contain (or equal) any of the sets* S_j.

We note that this readily includes all the "usual" examples of mutual exclusion problems for which the number of participating processes is finite and fixed in advance. We do not consider problems with unknown numbers of processes or unlabelled processes. E.g., if of n processes only one may enter its critical section at a time, the minimal mutually excluding sets (MMESs) are all pairs of processes; if k may enter at a time where $k < n$, the MMESs are the sets of $k+1$ processes. For a readers-writers problem, every set containing two writers, or one writer and one reader, is a MMES. For further justification of this definition, see [13].

By a *mutually excluding set* of a *mutual exclusion problem* $\mathcal{P} = (\mathrm{PR}, S)$ we mean any set of processes containing a minimal mutually excluding set. A *nonexcluding set* is a set of processes not containing such a set.

By a *simple solution* to a mutual exclusion problem we mean a collection of entry and exit sections (entry and exit protocols) which satisfy

1. *Mutual exclusion.* No mutually excluding set of processes will all be in their critical sections at the same time;
2. *Deadlock avoidance.* Eventually some process will enter its critical section;
3. *Possible entry.* For any nonexcluding set of processes there exists a schedule such that all these processes can be eventually in their critical sections at one time.

We do *not* demand fairness or even starvation avoidance. That is, we accept as a simple solution one meeting our three conditions even though some particular process may never enter its critical section.

By a *fair solution* of a mutual exclusion problem, we mean a collection of entry and exit sections constituting a simple solution and meeting the additional condition

4. *Eventual entry.* Every process which enters its entry section will eventually enter its critical section.

Of course we will later consider further conditions such as bounded waiting.

Restrictions. In this paper we drastically restrict communication methods between processes in their entry and exit sections. In particular, we rule out the reading and writing of shared variables and standard message-passing methods. In fact, the **only** commands we allow in entry and exit sections are the two commands $\mathbf{P}(s)$ and $\mathbf{V}(s)$ where $\mathbf{P}$ and $\mathbf{V}$ are the standard operations on semaphores.

Definitions. For other definitions of **semaphores**, see [1]. When we speak of processes as sharing a semaphore s of maximum value k with initial value j where k and j are integers and $0 \leq j \leq k$, then processes may include in their entry and exit sections the commands $\mathbf{P}(s)$ and $\mathbf{V}(s)$ which behave as if there were a shared variable s, $s \geq 0$ initialized to j and the commands were programmed atomically as follows:

$\mathbf{P}(s)$ means: wait until $s > 0$, then $s = s - 1$ and proceed.

$\mathbf{V}(s)$ means: $s = s + 1$ and proceed.

We call a semaphore *binary* if $k = 1$. We assume binary semaphores are initially 1 unless we specify otherwise. Note that our $\mathbf{V}(s)$ operation is non-blocking and does not enforce the maximum k; we leave keeping it below the maximum to the programmer.

We call a semaphore *weakly fair* if it meets the following condition: Suppose a process PR[i] executes $\mathbf{V}(s)$ and then $\mathbf{P}(s)$. If at least one other process was blocked and waiting in $\mathbf{P}(s)$ when PR[i] executes $\mathbf{V}(s)$, then some process other than PR[i] will be allowed to complete its $\mathbf{P}(s)$ before PR[i].

Note that our definition of "weakly fair semaphores" is different than in most temporal logics ([4], [6], [2]), where weakly fair is defined by stating that if a transition is continuously enabled, it will be eventually taken. Our definition is weaker and guarantees only that another enabled process than a current one will be executed. There is a significant literature on weakly fair

section of PR[x] with the following sequence of commands: (here j is an abbreviation for $\{x, y, z\}$)

$$\mathbf{P}(s_{j,\{x,y\}}); \mathbf{P}(s_{j,\{x,z\}}); \mathbf{P}(s_j); \mathbf{V}(s_{j,\{x,y\}}); \mathbf{V}(s_{j,\{x,z\}});$$

and similarly in PR[y] and PR[z]. That is, we apply exactly the one–of–t solution that we derived in Section 2 above to ensure that exactly one process at a time will attempt to execute the command $\mathbf{P}(s_{\{x,y,z\}})$. If the new semaphores ($s_{j,\{x,y\}}$ and so on) are weakly fair, this has the result of giving fair access to $\mathbf{P}(s_{\{x,y,z\}})$

We do this for each semaphore which is called for by more than two processes. We need to show that the resulting set of programs provides not only a simple solution, but a fair solution, to $\mathcal{P}$.

Example 4.2. Example of fair mutual exclusion using all pairs.

For a concrete illustration, consider four processes PR[1] to PR[4]. $s = s_{\{1,2,3,4\}}$ will be the semaphore shared by all four; the program below is the same whether this is a 1–of–4 , 2–of–4, or 3–of–4 semaphore. To "protect" access to s we introduce 6 semaphores which we denote by $s_{\{1,2\}} \cdots s_{\{3,4\}}$ (which are abbreviations for $s_{\{,1,2,3,4\},\{1,2\}} \cdots s_{\{,1,2,3,4\},\{3,4\}}$.) The essential parts of the four programs are as follows:

	PR[1]	PR[2]	PR[3]	PR[4]
entry :	$\mathbf{P}(s_{\{1,2\}})$	$\mathbf{P}(s_{\{1,2\}})$	$\mathbf{P}(s_{\{1,3\}})$	$\mathbf{P}(s_{\{1,4\}})$
	$\mathbf{P}(s_{\{1,3\}})$	$\mathbf{P}(s_{\{2,3\}})$	$\mathbf{P}(s_{\{2,3\}})$	$\mathbf{P}(s_{\{2,4\}})$
	$\mathbf{P}(s_{\{1,4\}})$	$\mathbf{P}(s_{\{2,4\}})$	$\mathbf{P}(s_{\{3,4\}})$	$\mathbf{P}(s_{\{3,4\}})$
	$\mathbf{P}(s)$	$\mathbf{P}(s)$	$\mathbf{P}(s)$	$\mathbf{P}(s)$
	$\mathbf{V}(s_{\{1,2\}})$	$\mathbf{V}(s_{\{1,2\}})$	$\mathbf{V}(s_{\{1,3\}})$	$\mathbf{V}(s_{\{1,4\}})$
	$\mathbf{V}(s_{\{1,3\}})$	$\mathbf{V}(s_{\{2,3\}})$	$\mathbf{V}(s_{\{2,3\}})$	$\mathbf{V}(s_{\{2,4\}})$
	$\mathbf{V}(s_{\{1,4\}})$	$\mathbf{V}(s_{\{2,4\}})$	$\mathbf{V}(s_{\{3,4\}})$	$\mathbf{V}(s_{\{3,4\}})$
(C.S.)	C.S.	C.S.	C.S.	C.S.
exit :	$\mathbf{V}(s)$	$\mathbf{V}(s)$	$\mathbf{V}(s)$	$\mathbf{V}(s)$

The argument that this is fair is essentially identical to the argument leading to Theorem 2.2. One of the n processes here gets its chance at $\mathbf{P}(s)$ after at most 2^{n-1} turns by other processes. Once it is the unique process waiting at $\mathbf{P}(s)$, it is guaranteed entry to the critical section as soon as the semaphore s becomes available, which will be no later than the next time a process completes its exit section.

We pay a price for the use of the full tournament guard to get a fair solution. No entry section can proceed fully in parallel with another entry section now; much more parallelism was possible when only semaphore s was used (without the guard). Of course, the delay incurred may be relatively small if the number of semaphore operations is not too large and the time to execute semaphore operations is small compared to the time in critical sections.

Adding all pairs of binary semaphores is necessary, because weakly fair semaphores "remember" only which semaphore performed the last successful **P** operation. If we had available "fair" semaphores for n processes, which guaranteed entry to each waiting process before allowing any one to enter a second time, the binary semaphores would not be needed.

Theorem 4.1. *There is a fair solution to any mutual exclusion problem which depends only on simple use of (not necessarily fair) initially-k and weakly fair binary semaphores.*

Proof:
We introduce initially-k semaphores for each minimal mutually excluding set, just as in Theorem 3.1. Then we add a full tournament of weakly fair binary semaphores to protect each of those semaphores which is not already a weakly fair binary semaphore. We need to check the four basic conditions for a fair solution.

(1)*Mutual exclusion.* This is exactly as in the prior section: each process still calls on the semaphores that admit at most $t-1$ of the t elements in the minimal mutually excluding set.

(2)*Deadlock avoidance.* Since we use the lexicographic order of subscripts in each mini-mutual exclusion problem (for access to the semaphores based on sets of size greater than two), and none of the newly introduced binary semaphores occurs in more than one of the mini-problems, none of these mini-problems deadlock. Thus we can treat each expanded section of the algorithm as if it were unexpanded, and the lack of circular wait there still shows that the global algorithm does not deadlock.

(3)*Possible entry.* As before, processes in a non-excluding set can enter their critical sections by passing their entry sections one at a time.

(4)*Eventual entry.* We must show that every process will eventually enter its critical section. It will suffice to show that each process will eventually pass each $\mathbf{P}(s_j)$ in its entry section. With respect to any $\mathbf{P}(s_j)$ where s_j is a set of two elements, this is easy to see: the second process to reach such a point will pass it when the first process performs the appropriate $\mathbf{V}(s_j)$ in its exit section. To see that a process will pass the expanded section around each $\mathbf{P}(s_j)$ where s_j is a set of more than two elements, observe that all processes competing for $\mathbf{P}(s_j)$ now engage in a mutual exclusion algorithm so that only one requests it at a time, and this algorithm is fair, that is, each process is guaranteed a chance to be, eventually, the only process executing $\mathbf{P}(s_j)$. Thus, if it must wait, it will be the only process waiting when some other process performs a $\mathbf{V}(s_j)$ and it will get the opportunity to complete its $\mathbf{P}(s_j)$, as desired, before any other process is allowed to attempt a $\mathbf{P}(s_j)$. □

There is an alternative solution which, in some cases, may shorten the entry section at some further cost in parallelism. Instead of placing a full-tournament guard around each individual $\mathbf{P}(s_j)$, place a single full-tournament guard (on the full set of processes) around each entire entry section. The effect is to demand a lock on all the s_j semaphores (the semaphores actually representing the minimal mutually excluding sets) before asking for any of them. This will still be a fair solution, but will destroy a great deal of parallelism. If the mutually excluding sets, for example, are $\{1,2\}$ and $\{2,3\}$ and the processes attempt to enter in order $1,2,3,4$, then process

critical section) on condition that it is never true that all interest groups have a member present (in the critical section).

Clearly, the minimal mutually excluding sets consist of exactly k members, one from each group. Thus the number of such sets is m^k, and each process is a member of m^{k-1} minimal mutually excluding sets.

The straightforward solution above calls for creating m^k semaphores, one for each minimal mutually excluding set, each with an initial and maximum value of $k-1$. This would prevent all members of any minimal mutually excluding set from entering, as desired. It would however require each process to request m^{k-1} semaphores before entering its critical section, even for a simple solution, and the fair solution precedes each of those **P** operations with $k-1$ others for a total of km^{k-1} **P** operations in each entry section.

Can this number of **P** operations be substantially reduced? In [14] it is shown that it cannot be, even using larger initial values for the semaphores or multiple calls to a single semaphore by the same process. That paper does rely on the fact that we are using only simple semaphores, leaving open whether there is an improvement using split semaphores or some other variation.

The key idea of the proof is as follows: consider the exclusion problem enforced by a single semaphore. Each process wanting (one or more instances of) that semaphore can be assigned an integer: how many times it wants that semaphore while in its critical section. Any processes whose values sum to more than the maximum value of that semaphore must contain a minimal mutually excluding set. By an argument analogous to arguments on excluded subgraphs of threshold graphs, no minimal mutually excluding set constructed in this way can intersect more than one interest group in a set of more than one element. Hence, in the particular configuration of this problem, no one semaphore can enforce mutual exclusion for more than m minimal mutually excluding sets.

By contrast this problem is quick and easy to solve (a simple solution, not a fair solution) using shared variables. Loosely, place a blackboard outside the meeting room, with an integer available to show the number of members of each group present and one to show how many groups are now represented. A member wanting to enter the room may enter if her group already has a member in, incrementing the count for her group. If she is the first of her group to arrive, she increments the number of groups if it will still be less than k and sets her group's counter to one. A member leaving decrements her group's counter and, if this makes it zero, decrements the number of groups.

This may be modified to make a fair solution using common techniques; something as simple as a waiting queue will suffice.

6. Thoughts on complexity

In a mutual exclusion problem, if all minimal mutually excluding sets are of size two, there are at most $n(n-1)/2 = O(n^2)$ such sets for any collection of n processes, so at most $O(n^2)$ binary semaphores are required and no process needs to access more than $n-1$ of them. If these binary

semaphores are weakly fair, each process will be able to enter its critical section following at most $O(n^2)$ entries by competitors.

We may regard the situation above as a graph with processes as vertices and an edge for each mutually excluding pair. Using the methods of [11] it is clear that one semaphore per maximal clique in this graph will suffice to enforce mutual exclusion, and there are at most $n^2/4$ such cliques. (The so- called "unisex bathroom" problem [1] illustrates this point well, although there it is still true that the minimal mutually excluding sets are of size two.) Since however the semaphore associated with a clique of order greater than two will be wanted by more than two processes, a system of protecting weakly fair binary semaphores may be needed to ensure fairness. Thus, unless the cliques are relatively small and relatively nonoverlapping, using them may not be an improvement.

The situation for mutual exclusion problems where minimal mutually excluding sets may have more than two elements is substantially different. The excluded taxpayer problem shows that in general even a simple (not necessarily fair) solution to the mutual exclusion problem may require exponentially many semaphores, exponentially long entry sections, and hence exponentially long time for any process to enter its critical section even given the best possible schedule. This shows that it is not at all reasonable to actually propose using only **P** and **V** operations instead of shared variables, even in circumstances when it is actually possible.

In general, if a mutual exclusion problem has m minimal mutually excluding sets, each of k members with $k > 2$, simple mutual exclusion by the methods above will require $O(m)$ semaphores and fair mutual exclusion $O(mk^2)$ semaphores. If a typical process belongs to only $j < m$ of the minimal mutually excluding sets, its entry section will need $O(jk)$ **P** operations. The delay to enter its critical section may still depend on the global situation, however, just as in one standard dining philosophers solution, one philosopher may have to wait while all the others eat consecutively.

7. Concluding remarks

Given a very general mutual exclusion problem, in the form of a set of labelled processes and a set of minimal mutually excluding processes, the methods of this paper automatically produce a mutual exclusion algorithm and a fair mutual exclusion algorithm using no communication primitives except simple semaphores. For the fair solutions, the binary semaphores are required to be weakly fair.

For many of the classical problems, and for mutual exclusion problems where the size of the minimal mutually excluding sets is two, or where the number of mutually excluding sets involving a given process is linear, the solutions constructed here generally run in time quadratic in the number of processes. It seems that in many cases this can be reduced but as yet we do not have a general method to do so.

There exist cases, such as the excluded taxpayer problem, where the solutions constructed here become exponentially large even though the problem can be easily solved using other

methods. Hence the method proposed here does not seem suitable for "automatic" use, e.g. in a parallelizing compiler, without further study.

References

[1] Andrews G.R., *Concurrent Programming: Principles and Practice*, Benjamin/Cummings, 1991.

[2] Francez N., *Fairness*, Springer-Verlag, 1986.

[3] Janicki R., Lauer P.E., *Specification and Analysis of Concurrent Systems: The COSY Approach*, Springer-Verlag, 1992.

[4] Lamport L., *'Sometime' is sometimes 'Not Never': On the Temporal Logic of Programs*, Proc. 7th ACM Symp. on Principles of Programming Languages, (1980), 174-185.

[5] Lynch N.A., *Distributed Algorithms*, Morgan Kaufmann, 1996.

[6] Manna Z., Pnueli A., *Verification of Concurrent Programs: The Temporal Framework*, in: R.S.Boyer,J.S.Moore (eds) The Correctness Problem in Computer Science, Academic Press, 1981, 215-273.

[7] Manna Z., Pnueli A., *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1992.

[8] Martin A.J., *A new generalization of Dekker's algorithm for mutual exclusion*, Info. Proc. Letters 23 (1986), 295-297.

[9] Martin A.J. and Burch B.R., *Fair mutual exclusion with unfair P and V operations*, Info. Proc. Letters 21 (1985), 97-100.

[10] Morris J.M., *A starvation-free solution to the mutual exclusion problem*, Info. Proc. Letters 8 (1979), 76-80.

[11] Ordman E., *Threshold coverings and resource allocation*, Proc. 16th Southeastern International Conf. on Graph Theory, Combinatorics, and Computing, Congr. Numer. 49(1985), 99–113.

[12] Ordman E. *Dining philosophers and graph covering problems*, J. Combinatorial Mathematics and Combinatorial Computing 1(1987), 181–190.

[13] Ordman E., *Cliques in hypergraphs and mutual exclusion using tokens*, J. Combinatorial Math. and Combinatorial Computing 19 (1995), 209-224.

[14] Ordman, E., *A mutual exclusion problem needing exponentially many tokens*, University of Memphis Combinatorics Preprint Series, Memphis, Tennessee, USA, April, 1998.

[15] Udding J.T., *Absence of starvation using weak semaphores*, Info. Proc. Letters 23 (1986), 159-162.

[16] Tanenbaum A., *Modern Operating Systems*, Prentice-Hall, 1992.