Randomized Mutual Exclusion Algorithms Revisited*

Eyal Kushilevitz† Michael O. Rabin‡

Abstract

In [4] a randomized algorithm for mutual exclusion with bounded waiting, employing a logarithmic sized shared variable, was given. Saias and Lynch [5] pointed out that the adversary scheduler postulated in the above paper can observe the behavior of processes in the interval between an opening of the critical section and the next closing of the critical section. It can then draw conclusions about values of their local variables as well as the value of the randomized round number component of the shared variable, and arrange the schedule so as to discriminate against a chosen process. This invalidates the claimed properties of the algorithm.

In the present paper the algorithm in [4] is modified, using the ideas of [4], so as to overcome this difficulty, obtaining essentially the same results. Thus, as in [4], randomization yields simple algorithms for mutual-exclusion with bounded waiting, employing a shared variable of considerably smaller size than the lower-bound established in [1] for deterministic algorithms.

*Research supported by research contracts ONR-N0001491-J-1981 and NSF-CCR-90-07677.

†Aiken Computation Lab., Harvard University and Computer Science Dept., Technion. e-mail: eyalk@das.harvard.edu .

‡Aiken Computation Lab., Harvard University and Institute of Mathematics, Hebrew University of Jerusalem. e-mail: rabin@das.harvard.edu .

PoDC '92-8/92/B.C.
© 1992 ACM 0-89791-496-1/92/0008/0275...$1.50

1 Introduction

In this paper we deal with the well-known *mutual-exclusion* problem: Let $P_1, \cdots, P_N$ be N processes that from time to time need to execute a critical section in which exactly one is allowed to employ some shared resource. They can coordinate their activities by use of a shared *test-and-set* variable v (i.e., testing and setting v itself is an atomic action, and access to v is always available to a P_i scheduled to do so). This problem was suggested by Dijkstra [2] and was discussed in many papers since then (see, for example, [1, 4] and the literature cited there). A solution for this problem is an algorithm that guarantees freedom from *deadlock* (this alone can be achieved by the use of a one-bit semaphore) and freedom from *lockout*.

Burns et. al. [1] considered the following question: What should be the size of the shared variable v so that (deadlock-free, lockout-free) mutual-exclusion can be implemented? This question is not only of theoretical interest but also of practical interest. This is because in practice test-and-set is not an atomic operation and what we really assume is that reading the variable and immediately writing it can be done very fast so that

no other operations will interrupt it. Such an assumption is reasonable only for "short" variables. Burns et. al. [1] proved that if *deterministic* algorithms are used then an $\Omega(\log N)$-bit shared variable is required, and in fact this number of bits is also sufficient.

Rabin [4], presented a *randomized* solution for the problem using an $O(\log \log N)$-bit shared variable. His algorithm was based on the following lemma:

> For *any* $1 \leq m \leq N$ processes, say $P_1, \cdots, P_m$, if each P_i randomly, with a *geometric* distribution, draws a number $1 \leq b^{(i)} \leq \log N + 4$, then with probability at least $2/3$ $b^{(j)} = \max b^{(i)}$ will hold for exactly one j.

The algorithm works in rounds, where a round is defined to be the time between two successive entrances to the critical section. To explain the algorithm we provisionally assume that the shared variable v contains a field, updated by the process entering the critical section, representing the current round number r. In addition the shared variable contains a flag to indicate whether the critical section is close or open, and a field b that contains the maximum number drawn during this round. Each process trying to enter the critical section during round r, upon accessing the shared variable, draws a number according to the geometric distribution, and updates the field b in the shared variable by the maximum among his number and the current value of b. If it already drew a number in round r it does *not* draw a number again. If the critical section is open and the number it drew equals b (the maximal number drawn in this round), it enters the critical section and starts a new round.

This solution is not only deadlock-free and lockout-free but also satisfies (using the above lemma) a powerful fairness property: If a process P_i participates in a trying round together with m other processes, it has a probability of $\Omega(1/m)$ to enter the critical section at the end of this trying round. This property is called in [4] *bounded waiting*. It was also shown, based on an idea of Ben-Or, that this algorithm can be modified so as to use just a constant size shared variable. This version, however, guarantees only a weaker fairness property: If a process P_i participates in a trying round, it has a probability of $\Omega(1/N)$ to enter the critical section at the end of this trying round. This is a much weaker property since in practice m, the number of processes competing for the critical section during a trying round, is typically much smaller than N, the number of processes in the system. It is important to remark that in both versions of the algorithm deadlock is never possible.

The difficulty with these algorithms is that we assumed that the unbounded round number r is part of the shared variable v. (All the other fields of v are of the appropriate size.) The idea for dealing with this problem was to replace the use of the round number r by a *randomized round number* (i.e., a random bit chosen by the process entering the critical section). The intuition was that a randomized round number is enough to guarantee that a process will not draw a number more than once at a trying round, and it seemed that the probability that a process will not draw a number at all is exactly $1/2$. Therefore, the same analysis seemed to work.

Recently, Saias and Lynch [5] showed that this is not true. They presented some examples in which an adversary scheduler can lockout a process P_i. Summarizing these examples there are two problems with the use of randomized round number instead of the actual round number in the above algorithms:

- Processes that lost in one trying round (i.e., did not enter the critical section) may remain with "high" lottery numbers. Such an occurance can be observed by the adversary according to the external behavior of the processes (without looking at the content of their local variables nor at the shared variable). Later, these processes can participate in a trying round which has the same randomized round number, together with the process P_i. Therefore, they will not draw new numbers (and remain with the old "high" numbers) and hence the probability of P_i to win the lottery in such a case is smaller than it should be.

- The adversary can learn whether the current randomized round number equals the randomized round number in the last trying round P_i participated in, by observing the external behavior of other processes participated with P_i in that previous round. Then, the adversary can schedule P_i only in rounds with same randomized round number. This will cause P_i not to draw a new number and this again decreases the probability of P_i to win the lottery.

We modify the algorithms presented in [4] using shared variables with the same number of bits (up to a constant) achieving the same fairness properties. The key ideas for overcoming the above two problems are:

- The modified algorithms make sure that processes which lost in a lottery will not keep high lottery numbers that can be used in future rounds.

- In the modified algorithms, each trying round is divided into two parts: the *drawing round* in which the processes draw their numbers, and the *notification round* in which the processes that took part in the drawing round find out who win and who lost. The idea is that during the drawing round, the external behavior of processes does not depend on the outcome of the lottery, and therefore during this time the adversary cannot gain information about the contents of the shared variable nor the local variables of the processes. During the notification round, the lottery is already over and the winner and the losers are already determined.

We believe that the separation idea may be useful in the design of other randomized protocols. To summarize: randomization yields simple algorithms for mutual-exclusion with bounded waiting, employing a shared variable of considerably smaller size than the lower-bound for deterministic algorithms.

2 The Modified Algorithm

2.1 Definitions and General Plan

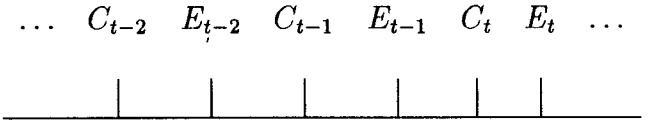
Let $P_1, \cdots, P_N$ be the N processes in the system. The processes coordinate their activities by use of a shared test-and-set variable $v = (b_{even}, r_{even}, b_{odd}, r_{odd}, s, p, w, partic)$ where each b component is $B = \log_2 N + 4$-valued, and the other components are $0, 1$ valued. Each process P_i has local variables $b^{(i)}, r^{(i)}, d^{(i)}$ and some flags. Variables local to P_i are denoted $x^{(i)}$.

The processes will use a geometric distribution to pick numbers between 1 and B. Namely, each $1 \leq i \leq B-1$ is drawn with probability $\frac{1}{2^i}$ and B is drawn with probabil-

ity $\frac{1}{2^{B-1}}$. We shall denote the act of drawing such a number by $b^{(i)} := randoml$. It was shown in [4] that for any $1 \leq m \leq N$ processes, say $P_1, \cdots, P_m$, if each P_i randomly, with a geometric distribution, draws a number $1 \leq b^{(i)} \leq B$, then with probability at least $2/3$ $b^{(j)} = \max b^{(i)}$ will hold for *exactly* one j.

During the computation, each process P_i is in one of four possible phases: *Trying phase*, in which it attempts to enter the critical section, *Critical phase*, in which it executes the critical section, *Exit phase*, in which it leaves the critical section, or *Remainder phase*, in which it does local computations. In this paper, we assume the same adversary scheduler that was postulated in [4]. Namely, at any given time the adversary scheduler can observe the *external behavior* of the processes (i.e., which of the four phases each process currently executes), and use this information (together with its information on the past behavior of the processes) to determine which process will be the next to access the shared variable. The adversary scheduler *cannot* observe the content of the shared variable nor the content of any local variable. More formally, let a *run* be a (finite or infinite) sequence $(i_1, x_1), \ldots, (i_k, x_k), \ldots$, where x_j indicates which phase process p_{i_j} started or whether it accessed the shared variable. A *scheduler* is a (probabilistic) function that on a finite run σ gives the name of the next process to access the shared variable. A run is *proper* if it satisfies the obvious consistency conditions.

The whole computation by $P_1, \cdots, P_N$ leading to entrance into the critical section is organized in intervals (the E_t, C_t are *logical*, not physical, times).



where $[C_t, E_t)$ is the t-th critical section, with C_t (and $s := 1$) marking the entry, E_t (and $s := 0$) marking the exit from this section.

In [4], the processes P_i trying to enter the next critical section $[C_t, E_t)$ drew, during the interval $[C_{t-1}, C_t)$, tickets $b^{(i)} := randoml$ and posted the result in v by $b := \max(b, b^{(i)})$. During $[E_{t-1}, C_t)$ the first process to arrive with a highest ticket $b^{(i)} = b$ enters the critical section. The crucial modification in the present algorithm is the following. In the t-th *drawing round* $[C_{t-2}, C_{t-1})$, participating processes draw numbers to determine the winning process P_i enabled to enter the t-th critical section at time C_t. By time C_{t-1}, all but at most B of the participants in the drawing round $[C_{t-2}, C_{t-1})$ will know that they lost in this drawing. In the t-th *notification round* $[C_{t-1}, C_t)$ each of the remaining participants finds out whether it won or lost, thus by time C_t there will be a P_i knowing that *it alone* won entrance to the critical section. The whole computation is arranged so that, for every t, the $t+1$-st drawing round overlaps with the t-th notification round, as indicated in Figure 1.

In other words, the interval $[C_{t-1}, C_t)$ serves both as the t-th notification round and as the $(t+1)$-th drawing round. To disambiguate this dual function of an interval $[C_{t-1}, C_t)$, the intervals are classified as even and odd. The parity component p of v will satisfy in the above interval $p = (t-1) \bmod 2$. Thus $p = 0$ signifies an even drawing round for entrance in the following even critical section at C_{t+1}, as well as an odd notification round to notify the winner in $[C_{t-2}, C_{t-1})$ - the just previous odd drawing round.

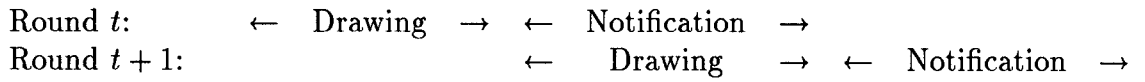


Figure 1: Organization of the computation in rounds

The key point is that since the drawing determining the winner P_i enabled to enter at C_t *ends* at time C_{t-1}, before the opening of the critical section at E_{t-1}, there will be no possibility for the scheduler to infer significant information about processes' local variables or about the r component of v while the drawing round $[C_{t-2}, C_{t-1})$ is in progress. It will also be seen that all participants losing in a drawing leave that drawing with their local lottery ticket = 0. Thus the scheduler cannot hoard processes with large ticket values and use them in later drawing rounds, as it could in the original version [4] of the algorithm.

2.2 Detailed Description of the Algorithm

When P_i is scheduled to test and set v, it knows by looking at local and global flags whether it should execute in some drawing round $[C_{t-1}, C_t)$ and, if yes, whether $t-1$ is even or odd. Assume P_i is in an even drawing round. It executes the protocol of Figure 2.

The test (1) in that protocol ensures that P_i will not draw more than once in $[C_{t-1}, C_t)$. If in the drawing it did not exceed the current value b_{even}, then it knows it lost and leaves with $d^{(i)} = b^{(i)} = 0$. Later it will not execute Notification, only Drawing.

Assume that $P_1, \ldots, P_k$ participated in $[C_{t-1}, C_t)$ and executed (2)-(3) (i.e., each of them at the time he drew his number was a local maxima). Then by C_t we have $b_{even} = d^{(1)} + \ldots + d^{(k)}$ (intuitively, $d^{(i)}$ is the contribution of P_i to b_{even}) and the unique winner is actually already determined. In particular, it follows that the number of processes executed (2)-(3) is small (in fact, it cannot be larger than B). All the other processes that participated in the drawing round already know they lost. This is very important because in the claimed size of v we cannot count, for example, the number of processes which participated in the lottery. The way we make sure that all the processes $P_1, \ldots, P_k$ will take part in the notification round is by letting each P_i to subtract its contribution $d^{(i)}$ from b_{even}. When all of them are notified we will have $b_{even} = 0$.

Notifying the winner and the losers amongst $P_1, \ldots, P_k$, is done in the even notification round $[C_t, C_{t+1})$. At time C_t we have $s = 0, p = 1$ (an odd drawing round) and $w = 0$ (winner not yet notified). The structure of the Notification round is as follows: the processes who lost are waiting until the winner is notified. Then, the losing processes are all notified and only then the winner is enabled to enter the critical section (if it is open). By looking at flags, every P_i knows whether for it $[C_t, C_{t+1})$ is a Notification round (or a Drawing round). It executes the protocol of Figure 3.

After P_i is enabled it must wait until the critical section will be open. It executes the protocol of Figure 4. The salient point in that protocol is that the round count number, r_{even}, is set to a *random* value. It is on this feature that the following proof of Lemma 1 rests.

Note that $partic = 1$ at C_{t+1} if and only if

Even Drawing for P_i ($p = 0$):

(1) If $[r^{(i)} \neq r_{even} \vee b_{even} = 0] \wedge [d^{(i)} = 0 \wedge w^{(i)} = 0]$ then (* P_i has not drawn in $[C_{t-1}, C_t)$ and is not participating in the current odd notification *)

begin

$r^{(i)} := r_{even}$

$b^{(i)} := randoml$

If $b_{even} < b^{(i)}$ then (* P_i is a local maxima *)

(2) $d^{(i)} := b^{(i)} - b_{even}$

(3) $b_{even} := b^{(i)}$

else (* P_i knows it lost *)

$b^{(i)} := 0$

end

Figure 2: Even Drawing Protocol

Even Notification for P_i ($p = 1$):

If $w = 0 \wedge b^{(i)} = b$ then (* P_i is the winner *)

begin

$w^{(i)} := 1$ (* P_i knows it won *)

$w := 1$ (* winner was notified *)

$b_{even} := b_{even} - d^{(i)}$

$d^{(i)} := 0$

end

If $w = 1 \wedge w^{(i)} = 0$ then (* P_i knows it lost *)

begin

$b_{even} := b_{even} - d^{(i)}$

$b^{(i)} := 0$ (* clean up local variables *)

$d^{(i)} := 0$

$r^{(i)} := nil$

P_i executes **Odd Drawing**

end

If $w^{(i)} = 1 \wedge b_{even} = 0$ then (* all losers were notified *)

P_i is **enabled** to enter the critical section

Figure 3: Even Notification Protocol

```
Enabled P_i Enters Even Critical Section:
If s = 0 then                                  (* critical section is open *)
begin
        w := 0; w^(i) := 0
        b^(i) := 0                             (* clean up local variables *)
        d^(i) := 0
        r^(i) := nil
        r_even := random(0,1)                  (* Assign 0 or 1 with equal probabilities *)
        p := 0                                 (* start of even Drawing *)
        s := 1                                 (* critical section closed *)
        If b_odd > 0 then                      (* some process drew in [C_t, C_{t+1}) *)
                partic := 1                    (* there was participation *)
        else
                partic := 0                    (* no participation *)
end
```

Figure 4: Entrance to Even Critical Section

some P_j drew in $[C_t, C_{t+1})$. The above protocols should be augmented by the provision that a trying process P_i accessing v, upon finding $partic = 0$, sets $partic := 1, w := 1, w^{(i)} := 1$ and is enabled to enter the next critical section. In the beginning, v is initialized with $partic := 0$.

2.3 Correctness

It goes without saying, that the adversary scheduler can always discriminate against a chosen P_i by consistently scheduling it very rarely or scheduling it together with many other processes. However, given that P_i participates in a drawing round with $m-1$ other processes, the following lemma gives a lower bound on the probability of P_i to enter the critical section which depends only on m, the *actual* number of participants in the drawing round (and not on N). It is important to note that this probability is also independent of the past.

Before formulating the lemma, we need to argue that the probability space that we are dealing with is well defined. We say that a (proper) run $\sigma = (i_1, x_1), \ldots, (i_k, x_k)$ is of *length* t if exactly t of the x_j's indicating a start of a Critical phase, and one of them is x_k. Let V_{t-1} be the *view* of the system at time C_{t-1}, just before r_{even} was randomly set to 0 or 1 (we assume that $t-1$ is even). That is, V_{t-1} consists of the values of the shared variable and all the local variables in the system at this time. Given σ, a (proper) run of length t, and a view V_{t-1} *consistent* with it (e.g., the process that enters the critical section at time C_t should be the winner of the previous drawing round $[C_{t-2}, C_{t-1})$ as reflected by V_{t-1}), the winner of the drawing round $[C_{t-1}, C_t)$ depends only on the random choices of the processes which are in a Trying phase during this time, and the random value of r_{even}. It is important to note that the process that will enter the critical section at time

C_t is already determined by V_{t-1}. The random choices made during the time $[C_{t-1}, C_t)$ affect the values of the shared variable and the local variables, but will affect the external behavior of the processes only at time C_{t+1} (until that time they all stay in the Trying phase). Therefore, the run σ is independent of the random choices made during this time, and hence we get a well defined probability space. (In a sense, we give here an additional power to the adversary by allowing him a total view of the system just before the drawing round starts.) Now, we can formalize the lemma:

Lemma 1: Let σ be any (proper) run of length t, and V_{t-1} be any view consistent with σ. If in σ process P_i participates in the drawing round $[C_{t-1}, C_t)$ together with $m-1$ other processes and if V_{t-1} is the view of the system at time C_{t-1}, then with probability at least $1/3m$ the process P_i will enter the critical section at time C_{t+1}.

Proof: Assume $t-1$ is even. At time C_{t-1} the value of the component r_{even} was randomly set to 0 or 1.

Clearly, all processes which do not participate in the drawing round $[C_t, C_{t-1})$ have no influence on the drawing (they either access only the fields of the shared variable connected with the odd drawing rounds or do not access the shared variable at all). It is also guaranteed by the algorithm that every process P_j that first joins the drawing has $b^{(j)} = 0$.

As claimed above, even though we allow the scheduler full information on the past (by giving him V_{t-1}), and in particular we allow him to look at $r^{(j)}$ of all P_j's, the run σ is independent of the value of r_{even}, and in particular is independent of whether $r^{(i)} \neq r_{even}$ at the time that P_i first access v during the drawing round. Hence, as r_{even} chosen to be 0 or 1 uniformly, if $r^{(i)} \neq nil$ then with probability $1/2$ we have $r^{(i)} \neq r_{even}$ when P_i first accesses v in $[C_{t-1}, C_t)$. In such a case P_i draws a new $1 \leq b^{(i)} \leq B$ in this round. (If $r^{(i)} = nil$ then P_i draws a new number in this round with probability 1.)

Every other process P_j of the m participating processes draws its random $b^{(j)}$ at most once, during the current drawing round. Those P_j's which do not actually draw have $b^{(j)} = 0$ and do not affect b_{even} at all. By Rabin's lemma (quoted in the Introduction), the probability of having a unique winner is at least $2/3$ (no matter what is the number of other processes that draw new numbers). Given that there is a unique winner then, by symmetry argument, each process that draws a new number has the same probability to be the winner. (Note that there is no symmetry in case that the winner is not unique; the process that draws the maximal number first is the winner.) All together, with probability at least $1/2 \times 2/3 \times 1/m = 1/3m$, process P_i will draw the maximum value in this drawing round and will be alone in this. Hence $b^{(i)} = b_{even}$ at time C_t and P_i must be the one to enter the next critical section at C_{t+1}. □

The property that for any $m \leq N$ if P_i is participating in a drawing round together with any $m-1$ other processes, then with probability $1/\gamma m$ it will win entrance to the critical section, is called in [4] *γ-bounded waiting*. Thus we have

Theorem 2 : Mutual exclusion with bounded waiting can be achieved for a N processor system by use of an $O((\log_2 N)^2)$-valued shared test-and-set variable.

This is Theorem 4 of [4] except that now we require a $(\log_2 N)^2$-valued variable, instead of $\log_2 N$ values. (In terms of the number of

bits, both the old and new theorems use an $O(\log \log N)$-bit variable.)

3 Concluding Remarks

Another version of the mutual exclusion algorithm given in [4] is based on a random drawing suggested by Ben-Or. Specifically, P_i draws the value 2 with probability $1/N$ and the value 1 with probability $1 - 1/N$. It is readily seen that if P_i participates in a drawing together with $m - 1$, $1 \leq m < N$, other processes, then with probability at least $1/2eN$ the process P_i will be the sole winner. Using this lottery we get

Theorem 3: For an appropriate constant c ($c \leq 3^2 \cdot 2^6$), starvation-free mutual exclusion for N processes can be achieved by use of a c-valued test-and-set variable.

This is the same result as in [4]. As in [4], Theorems 2 and 3 should be contrasted with the lower bound results in [1]. It is shown there that for *deterministic* algorithms an N-valued variable is necessary for mutual exclusion with bounded waiting (in the sense of [1]), and an $N/2$-valued variable is necessary for starvation-free mutual exclusion.

In fact, the last algorithm can be further simplified: in the Notification round all that is needed is for the unique winner to find out that he won. The numbers held by the losing processes are at most 1 and therefore if P_i participates in a drawing round then still with probability at least $1/2eN$ the process P_i will be the sole winner. Also, in the first algorithm b can be duplicated (thus, increasing the number of bits in v by a constant factor) so that the losing processes will not have to wait until the winner is notified. This may be helpful in scenarios where some of the processes are much "slower" than the others.

An interesting open problem is to try to show a better upper bound than the one presented in Theorem 2, or to prove a lower bound. A step in this direction was achieved in [3] where it is proven that there is no better lottery with the unique winner property. Therefore, if one tries to improve the upper bound he cannot hope to do that by using only a different lottery, but instead he should find a somewhat different algorithm.

References

[1] Burns, J. E., M. J. Fischer, P. Jackson, N. A. Lynch, and G. L. Peterson, "Data Requirements for Implementation of N-Process Mutual Exclusion using a single shared variable", *JACM*, Vol. 29, 1982, pp. 183-205.

[2] Dijkstra, E., "Solution of a Problem in Concurrent Programming Control", *CACM*, 321, 1966.

[3] Kushilevitz E., Y. Mansour, and M. O. Rabin, in preparation.

[4] Rabin, M. O., "N-Process Mutual Exclusion with Bounded Waiting by $4 \log_2 N$-Valued Shared Variable", *JCSS*, Vol. 25 (1), 1982, pp. 66-75.

[5] Saias, I., and N. Lynch, "Proving Probabilistic Correctness Statements: The Case of Rabin's Algorithm for Mutual Exclusion", This Proceedings.