SYMMETRIC AND ECONOMICAL SOLUTIONS TO THE MUTUAL EXCLUSION PROBLEM IN A DISTRIBUTED SYSTEM

Shimon COHEN and Daniel LEHMANN
Institute of Mathematics and Computer Science, Hebrew University, Jerusalem, Israel

Amir PNUELI
Department of Applied Mathematics, Weizmann Institute, Rehovot, Israel

Abstract. The mutual exclusion problem in a distributed system, in which each process has a memory of its own, into which it has exclusive write privileges but from which others may read, is reconsidered. Symmetric solutions are looked for. It is shown that, though no such solution may be deterministic, there are probabilistic solutions. Different solutions are provided for two processes, and then a solution is proposed for any number of processes. The solutions offered are amenable to a formal proof of their correctness with a small effort. The solutions are correct even against a very well informed scheduler, unlike Rabin's probabilistic solution to the mutual exclusion problem in a centralized system. Some of the solutions are correct even against an evil scheduler that knows in advance the results of the future random draws, in sharp contrast with the algorithms of Lehmann and Rabin (1981). The solutions are economical: mutual exclusion between two processes may be achieved with variables capable of holding four different values (to be compared with Peterson and Fischer's three), mutual exclusion between $n$ processes may be achieved with variables capable of holding ten different values (to be compared with Peterson and Fischer's fourteen). All solutions have been attained by careful reasoning and not by an exhaustive computer search; they exhibit general principles of design that may be useful in solving other similar problems.

1. The mutual exclusion problem in a distributed environment

The mutual exclusion problem is a now classical problem in concurrent programming, first proposed by E.W. Dijkstra (see [3, 4, 6] for early work and [8, 15, 11] for recent work on this problem). For the notions of critical section, remainder, trying and exit sections the reader is referred to the papers quoted above.

Relative to the mutual exclusion problem, we define a lockout as a computation in which one of the processes wishes to enter its critical section, but will never do so. A deadlock is a computation in which, at some point, some process wishes to enter its critical section and no process ever enters its critical section beyond that point. A computation is said to exhibit overtaking bounded by $k$, if every process that wishes, at any time, to enter its critical section, gets access to its critical section before any other process gets to enter its critical section $k+1$ times.
We are interested in solving the mutual exclusion problem in a distributed environment as introduced in [8]. We assume the existence of $n$ processors, each containing its own memory unit. In each of those memory units, there is a special area that may be read (but not written) by any processor. Except for this special area, a processor has exclusive access to its own memory. Deterministic solutions to the mutual exclusion problem in a distributed environment have been proposed in [8], [15] and [11]. The quality of a proposed solution is assessed by reference to different criteria, among them the size of the special area of memory used by the processors for communicating, the speed with which an interested processor will be allowed to enter its critical section and the immunity of the system to the possible failure of a processor. We concentrate on the first two criteria. The best solution proposed so far is that of [11], where a solution is proposed for two processors that requires a special memory area capable of holding three different values (this number is also shown to be a lower bound), and a solution for $n$ processors requiring an area capable of holding fourteen values. The solution also guarantees bounded waiting time.

2. Symmetric solutions

The solutions mentioned above are not symmetric, i.e., either the different processors follow different routines or the initial values of the memories of the different processors are not the same. Nevertheless, one expects a solution not to favour one of the processors among its competitors. This requirement of symmetry has been first formulated by Dijkstra and an up-to-date study may be found in [2].

A very simple symmetry argument can show that many problems do not have a deterministic symmetric solution (see, for example, [9] and [10]). The centralized version of the mutual exclusion problem has a symmetric deterministic solution. The distributed version does not.

There is wide agreement as to the necessity of behavioural symmetry, but not quite, yet, general agreement about the use of totally symmetric solutions. Symmetry is aesthetically pleasing (and that is important), but even more important is the fact that symmetric solutions automatically ensure behavioural symmetry (thus short-cutting a possibly delicate proof) and that their proofs of correctness tend to be eased by the symmetry. Symmetric solutions should also be preferred for economy reasons. A non-symmetric solution to the mutual exclusion problem for $n$ processes, such as that of [11] though it requires a shared (for reading only) variable of only constant (independent of $n$) size, requires each process to somehow remember some kind of identity number of size $\log n$. A symmetric solution, such as ours, does not. Also, it is always easier to manufacture a system consisting of identical parts than a system consisting of a large number of different parts that have to be assembled in a specific fixed layout. The rest of the paper is devoted to studying probabilistic symmetric solutions.
3. Probabilistic solutions

The main notions concerning probabilistic solutions will be explained, especially the notion of a schedule, which may be found in [9]. The basic idea of all our solutions is to let all processes compete for the shared resource by drawing a random value and let the process that obtains the 'highest' random value enter its critical section. The losers of a competition will then compete between themselves. If there is a tie, the processes go through another competition. The two main problems that arise in implementing this idea are: to make sure that processes compare up-to-date results of random draws and not out-of-date values and to define precisely the group of processes competing, so as not to wait for results of draws of processes that are not interested in competing.

4. A first solution for two processes

At first, we shall present a solution for the mutual exclusion problem between two processes, that guarantees that no process has ever more than one turn to wait. The solution is not economical, since the private variable of each process must be able to hold nine different values. Its interest lies in its simplicity and the simplicity of its proof of correctness. This solution ensures mutual exclusion with certainty, absence of deadlock with probability one and bounded overtaking with certainty. This is much stronger than finite expected overtaking. The solution below guarantees absence of deadlock, with probability one, even against an evil clairvoyant scheduler that knows in advance the results of future random draws; it is the first example of such a 'robust' solution (the algorithms of [9], for example, do not enjoy this property).

Each process uses a private variable my on which it has exclusive write privileges and refers to the private variable of the competing process by hers. The basic commands are assignments and wait statements. The wait statement has to be understood as busy waiting: looking from time to time until hers is found in a favourable state, and then, branch depending on the value found. The general version of the wait statement is:

\[
\text{wait until condition and then goto label}
\]
\[
\text{or condition and then goto label}
\]
\[
\text{or ...}
\]
\[
\text{endwait.}
\]

An absent goto part means go to the textually next statement.

Nothing is assumed concerning the rate at which checking is performed, except that it is performed an infinite number of times or until successful. The variables my and hers may take seven different values, in addition to the set of random values from which processes draw. Since this set of random values must contain at least two different values, the most economical (in space) version of our first algorithm
uses four-bits variables. We name those seven values: uninterested, interested, going-in, ready-to-draw, won, lost, tie. The initial value of the variables my and hers is uninterested. The set of values from which values are randomly drawn is totally ordered, and denoted Random.

Algorithm 1

Exit section:

30: my := uninterested

Trying section:

1: wait until
   hers in \{uninterested, interested, going-in\}
  endwait;
2: my := interested;
3: wait until
   hers = uninterested and then goto 4
   or hers in \{interested, ready-to-draw\} and then goto 6
  endwait;
4: my := going-in;
5: if hers = uninterested then goto Critical Section
else goto 6
fi;
6: my := ready-to-draw;
7: wait until hers in \{ready-to-draw\} ∪ Random endwait;
8: my := a random element of Random;
9: wait until
   hers = won v my < hers and then goto 10
   or hers = lost v my > hers and then goto 12
   or hers = tie v my = hers and then goto 14
  endwait;
10: my := lost;
11: wait until
   hers = uninterested and then goto Critical Section
  endwait;
12: my := won
13: wait until
   hers = lost and then goto Critical Section
  endwait;
14: my := tie;
15: wait until
   hers in \{tie, ready-to-draw\} and then goto 6
  endwait;
Note that the condition $my > hers$ holds only if both values are in the set $Random$. Otherwise, the condition is well defined and evaluates to false.

The proof of correctness proceeds the following way: First some invariant properties may be easily proved, in particular mutual exclusion. A first observation is that if one process is in its critical section with the value $lost$, then the competitor must be in its remainder section or at statement 1 (in both cases with value $uninterested$). There are only three entries to the critical section: statements 5, 11 and 13. If our process has entered its critical section from statement 5 or from statement 11, at entry time $my$ was $going-in$ or $lost$ (respectively) and $hers$ was $uninterested$, therefore the competitor was either at statement 1 or at statement 2; in any case it will not pass statement 3. If our process has entered its critical section from statement 13, at entry time $my$ was $won$ and $hers$ was $lost$. Therefore, the competitor was either at statement 11 or in the critical section. If it was at statement 11, it could not pass it. It could not be in the critical section with $hers$ equal to $lost$ due to our observation above. We proved that a violation of mutual exclusion may occur only as a consequence of a previous violation and therefore mutual exclusion is guaranteed.

The more interesting part of the proof concerns liveness properties. First let us show that no process will wait indefinitely in one of the $wait$ statements. Suppose we wait indefinitely at statement 1 (with $my$ equal $uninterested$). Then we can show, with the help of some invariants, that our competitor will, sometime, attain its remainder section. From then on, since we do not move, it will always stay in statements 1, 2, 3, 4, 5, critical section, 30 and remainder. At all times then its variable $hers$ will stay in the set $\{uninterested, interested, going-in\}$, and we shall test $hers$, find its value favourable and proceed to statement 2. Contradiction.

Suppose now that we wait indefinitely at statement 3 (with $my$ equal to $interested$). Invariant properties show that our competitor may only be at one of the statements: 1, 2, 3, 4, 5, 6, 7, critical section, 30 and remainder. It will then either stay indefinitely in its remainder section (with $hers$ equal to $uninterested$) or move to statement 7 and get stuck there indefinitely with $hers$ equal to $ready-to-draw$. In any case, after a certain time the variable $hers$ will stay indefinitely with a value that allows us to go on. Contradiction.

The reasonings concerning statements 7, 9, 11, 13 and 15 are very similar to the previous one.

It follows that a process that is interested in getting access to its critical section will eventually enter its critical section, unless it loops indefinitely in the only loop of the program: 6, 7, 8, 9, 14, 15. Invariant analysis shows that whenever a process is at statement 14, its competitor is at one of statements 9, 14, or 15. Our process will therefore move to 15, and to 6, but will not attain statement 6 before its competitor has attained at least 15, and at most 7. We see that our process may loop indefinitely only when its competitor also loops indefinitely and both processes keep in step: they draw the same number of times and compare (in statement 9) always freshly drawn values. Such looping may happen only as long as the two processes draw the same random value at each turn: this clearly has probability zero.
The maximum waiting time may be easily analyzed: as soon as our process executes statement 2, its competitor may enter its critical section, before our process does, at most once.

5. An economical algorithm

The previous algorithm used relatively large variables. Can we do better? A straightforward generalization of an argument of Peterson and Fischer [11] can show that no solution (even probabilistic and not symmetric) can be worked out, that uses variables capable of holding only two different values. We do not know whether there are symmetric solutions using variables capable of holding only three different values ([11] offers such a solution that is not symmetric). We propose a symmetric solution using four-values variables.

The basic idea is to use liberal policies regarding the synchronisation of the competition process: we shall allow competing processes to draw at very different rates and compare their random values to values drawn long ago by the competitor. Obviously, random values will be drawn from a set of two values only: \{high, low\}.

The algorithm we propose guarantees mutual exclusion with certainty (not with high probability), absence of deadlock with probability one and bounded overtaking with certainty. This obviously implies absence of lockout with probability one. Absence of deadlock, with probability one, is guaranteed against an evil scheduler that knows everything about the past. In comparison with the previous algorithm, the current one does not enjoy the robust feature of being deadlock free even against a clairvoyant scheduler. The possible values for the variables \textit{my} and \textit{hers}, in addition to the two random values already mentioned, are \{uninterested, interested\}. The first one is the initial value for both variables.

\textbf{Algorithm 2}

Exit section:

30: \textbf{if} \textit{my} = interested \textbf{then} goto 33
    \textbf{else} wait until \textit{hers} in \{uninterested, high, low\}
    \hspace{1em}and then goto 31
    \textbf{endwait}

31: \textit{my} := interested;
32: \textbf{wait until} \textit{hers} in \{uninterested, high\} \textbf{endwait};
33: \textit{my} := uninterested;
Symmetric and economical solutions to the mutual exclusion problem in a distributed system

Trying section:

1: wait until
   hers in \{uninterested, interested\}
endwait;
2. my := interested;
3: wait until
   hers = uninterested and then goto Critical Section
   or hers = interested and then goto 4
   or hers = high and then goto 8
endwait;
4: my := a random element of \{high, low\};
5: if my = high
   then
      wait until hers in \{low, uninterested\}
      and then goto Critical Section
      or hers = high and then goto 4
   endwait
   else
      if hers in \{uninterested, interested\} then goto 6
      elsif hers = low then goto 4
      else goto 9
   fi
   fi;
6: my := high;
7: wait until
   hers in \{uninterested, low\} and then goto Critical Section
endwait;
8: my := low;
9: wait until hers in \{uninterested, interested\} endwait;
10: my := high;
11: goto Critical Section;

Here we only present a sketch of the proof. No special problems occur in the proof of mutual exclusion.

One may then prove that our process has to wait indefinitely in a wait statement only with probability zero. As an example, let us prove that one cannot be stuck for ever in statement 9. Suppose we get stuck in statement 9, with value low forever. Invariant reasoning shows that, while we are at one of statements 8 or 9, our competitor cannot be at one of statements 8 or 9. If our competitor is at one of statements 11, 10, 7 or 6, it will eventually enter its critical section and then either move to statement 33 and the remainder section or move to statements 31, 32 and get stuck indefinitely in statement 32 with value interested. In this last case, we
should move and we have a contradiction. In the first case, our competitor may
either stay indefinitely in its remainder section with value uninterested, but in this
case we should move (a contradiction) or it will get to statement 1 and be stuck
there forever with value uninterested (a contradiction). If our competitor is at one
of statements 2 or 3, it will move to statement 3 and get stuck there with value
interested and we will move (a contradiction). If our competitor is at one of
statements 4 or 5, it will move to statement 5. If it draws high, it will enter its critical
section and we are in a case already treated. If it draws low, it will move back to
statement 4 and be given another chance of drawing high. With probability one,
our competitor will eventually draw high and move to its critical section.

The most delicate part of the proof is that our process cannot be looping
indefinitely in the only loop of the program: 4, 5. If it did, its value would, after a
certain time, always be high or low. The first step is to show that if our process is
at one of statements 4 or 5, with value high or low and its competitor is not at one
of statements 4 or 5, then our process will eventually, with probability one, leave
those statements and, by previous results, enter eventually its critical section.

Suppose, indeed, that our process is staying indefinitely in statements 4 and 5,
with value high or low, and that our competitor is not at one of statements 4 or 5.
By previous reasoning, our competitor will either

1. attain statement 9 and stay there until we move, or
2. attain its remainder section and stay there forever, or
3. attain statement 1 and stay there until we move.

In case (1) we shall find hers equal to low, and go on drawing until we eventually
draw high, and then enter our critical section. In cases (2) or (3) we shall find hers
equal to uninterested and move either to our critical section or to statement 6 and
then 7. At this point, our competitor would still be unable (or unwilling) to move
and therefore we would move to our critical section.

The crux of the proof is that, if both competitors are at one of statements 4 or
5, one of them will, with probability one, eventually leave those statements (by
previous results, the other one will eventually leave too). Suppose, indeed, that both
competitors are at one of statements 4 or 5, and none of them will ever leave those
statements. Clearly, both processes will draw an infinite number of times. Now we
wish to show that, whenever a process (call it A) draws a random value, there is a
fixed, positive probability that one of the competitors (A or B) will leave the loop
before A draws a second time.

Case 1: B is at statement 5. Here, if A draws the value (high or low) that is
different from that of B (and this event has probability 1/2), whoever will be next to
perform its own statement 5 will leave the loop.

Case 2: B is at statement 4. Here, if A draws the value that is different from that
of B (and this event has probability 1/2), then either A will be next to act, execute
statement 5 and leave the loop, or B will be next to act, and, by the analysis of
Case 1, somebody will leave the loop immediately with probability at least 1/2. It
follows that, in any situation, the probability that somebody will leave the loop,
before any further draw of $A$, is at least $\frac{1}{4}$. Since $A$ draws an infinite number of times, somebody leaves the loop, with probability one.

By a slight refinement of the proof above, one may see that as soon as process $A$ has performed statement 2, its competitor will not enter its critical section more than once before process $A$ does.

6. Mutual exclusion for $n$ competitors

We present now an algorithm that solves the mutual exclusion problem for $n$ processes in a distributed environment. Each process has a private variable $my$, that it can write into and reading privileges on the private variables of other processes. The private variables may take, in addition to at least two values used for random draws, the following values: \{uninterested, waiting, competing, goingin, lost, again, tie, breaktie\}. The initial value is uninterested. We use a slight generalisation of the wait statement used previously: \texttt{wait until all in Set} waits until all private variables of other processes are in Set. Its execution implies a repetition of simple waits. Thus the values of the variables belonging to other processes may be tested at different times, and we may decide on a positive answer while, in fact, the values never were in Set all at the same time. The reserved word some refers to any one of the private variables of the other processes; its use implies some hidden loop. The reserved word none is similarly understood.

Algorithm 3

Exit section:

30: $my := \text{uninterested}$;

Trying section:

1: $my := \text{waiting}$;
2: \texttt{wait until all in \{uninterested, waiting\}}
   \texttt{or some in \{goingin\}}
   \texttt{endwait};
3: $my := \text{competing}$;
4: \texttt{if some in \{lost, again, tie, breaktie\} \cup Random}
   \texttt{and none in \{goingin\}}
   \texttt{then goto 1;}
5: $my := \text{a random element of Random}$
6: \texttt{wait until all in \{uninterested, waiting, tie, lost\} \cup Random}
   \texttt{endwait;}
   \texttt{wait until}
all in \{\text{uninterested, waiting, tie, lost}\} \cup \text{Random}

\text{endwait};

7: if some > my then goto 17;

8: wait until
   all in \{\text{uninterested, waiting, tie, lost, my}\}
   \text{endwait};

9: my := tie;

10: wait until
    all in \{\text{uninterested, waiting, tie, breaktie, lost}\}
    \text{endwait};

11: if some in \{\text{tie, breaktie}\} then goto 15;

12: if some in \{\text{lost}\} then goto \text{Critical Section};

13: my := \text{goingin};

14: wait until
    all in \{\text{uninterested}\} \cup \text{Random}
    and then goto \text{Critical Section}
    \text{endwait};

15: my := breaktie;

16: wait until
    all in \{\text{uninterested, waiting, lost, breaktie}\} \cup \text{Random}
    and then goto 5
    \text{endwait};

17: my := lost;

18: wait until
    all in \{\text{uninterested, waiting, lost, again}\}
    \text{endwait};

19: my := again;

20: wait until
    all in \{\text{uninterested, waiting, again}\} \cup \text{Random}
    and then goto 5
    \text{endwait};

We give a brief indication of the ideas of the proof. A process may enter its critical section only from statement 12 or 14, after putting its variable to the value \text{tie} in statement 9, and checking that no one else has value \text{tie} in statement 11. This proves mutual exclusion.

The next step is to prove a number of invariant properties, showing that, essentially, processes proceed in an almost synchronised way, in the competition part, starting at statement 5.

Then one shows that the set of processes that take part in the competition (those at statements 5..20) is closed once the competition begins and that each one has a positive chance of entering its critical section. It follows that, with probability one, somebody enters its critical section, leaving a smaller set of competitors. The
last one of a competition to enter its critical section, goes through statements 13 and 14 and makes sure that all waiting processes enter the next turn of the competition. It is left to show that if nobody is in the competition, and somebody is waiting then somebody will enter the competition.

Acknowledgment

We are grateful to Sergiu Hart and Micha Sharir for discussions on the subject of probabilistic algorithms. We are indebted to Lenore Zuck for pointing out an error in the previous version of Algorithm 3, and for verifying the correct version.

References