1989

Proof of a Mutual Exclusion Algorithm-- A `Class'ic Example

Micha Hofri

Report Number:
89-913

PROOF OF A MUTUAL EXCLUSION ALGORITHM -- A 'CLASS'IC EXAMPLE

Micha Hofri

CSD-TR-913
October 1989

PROOF OF A MUTUAL EXCLUSION ALGORITHM – A 'CLASS'IC EXAMPLE

Micha Hofri†
Department of Computer Science, The Technion, Haifa, ISRAEL

1. INTRODUCTION

When we introduce the basics of concurrent processing in class, an important task is to convey to the students the inadequacy of our intuition in dealing with parallel operations and perceiving their interactions. This unpleasantness is handled in most texts by producing a sequence of "solutions" to the mutual exclusion (ME) problem, and pointing out their failure modes.

However, when finally a correct solution is presented, the correctness is rather claimed than proved; in particular, the students are not supplied with insight into the reason this solution is better than the earlier attempts, at least, not such that they could use susbsequently in similar programming problems. It is rather a negative situation, in that they fail to find in the correct protocol any bug, loophole... Some texts (and teachers) provide a 'hand waving' proof, some promise the existence of a methodology within which better proofs are hatched, and direct the students to (graduate level) courses such as "program verification", "programming logics", etc.

In this note, I present a proof of the n-processors protocol invented by Peterson (1981). I have used it several times in class, and it was my impression that this rather detailed proof contributed to the elucidation of the notions of 'atomicity' (or 'primitivity') and bounded delay. An important part of the proof is the unending effort to make the naturally descriptive language we use to state temporal relationships and the behavior of software sufficiently precise to provide what the students will recognize to be a valid mathematical proof. In particular, no new reasoning tools are presented beyond what the students had presumably encountered in basic mathematics courses, and in particular – induction over the integers.

Section 2 presents the protocol, Section 3 the proof. Some additional discussion concludes our treatment.

2. PETERSON'S n-PROCESS PROTOCOL

A set of n concurrent processes is denoted by p_1 to p_n. They all wish to access a critical resource, while maintaining ME. They do this via the protocol presented below. Two global arrays are used, q and *turn*. The initial values of the n elements of q and the $n-1$ elements of *turn* are all 0.

† Currently at the Department of Computer Science, Purdue University, W. Lafayette, IN 47907.

The only further assumptions are that the processes maintain private variables j and k, that are used as array indices, and that only load and store instructions are completed undisturbed. In particular, each process could be interrupted any number of times during the execution of each line in the protocol presented below, since even the most elementary ones, such as lines 4 or 5, involve the evaluation of array indices.

Alternatively, we may assume that each of $\{p_i\}$ has a dedicated processor, and they all operate asynchronously, up to ordering forced by the (common) storage controller. We use the standard infinitely-repeating format:

```
0.  global integer arrays $q[n]$, $turn[n-1]$
1.  $p_i$:     repeat forever
2.      {    non-critical part of $p_i$
3.           for $j=1$ to $n-1$ do
4.           {   $q[i]=j$
5.               $turn[j]=i$
6.           $L$:   for $k=1$ to $i-1$, $i+1$ to $n$
7.                     if ( ($q[k] \geq j$) and ($turn[j]=i$) ) goto $L$
8.           }
9.           $q[i]=n$; Critical section of $p_i$    (CS$_i$)
10.        $q[i]=0$
11.     }
```

Lines 3–8 are the entry protocol; line 10 is the exit protocol. An equivalent formulation of lines 6–7 could be used:

(6-7)′ wait until ((for all $k \neq i$ $q[k] < j$) or ($turn[j] \neq i$)).

3. PROOF OF THE PROTOCOL CORRECTNESS

We show that the above protocol provides

(a) mutual exclusion,

(b) deadlock freedom,

(c) no starvation, and

(d) with certain liveness assumptions – linear waiting.

The value of the local variable j of p_i is defined unless p_i is in its non-critical phase. It is convenient to call this value the "stage" of the protocol at which p_i executes. Since we want the notion of stage to be global, we say that when p_i is in its entry protocol, it is at stage $q[i]$, and when it enters CS$_i$ it passes to stage n (accordingly we added the statement $q[i]=n$ in line 9. Think of it as a metastatement, existing only to simplify the language used in the proof: it is not part of the original protocol and—as the proof below shows—is unnecessary for its correctness).

Also, if $q[i_1] > q[i_2]$, we say that p_{i_1} *precedes* (or is in front of) p_{i_2}.

The desired properties will be shown to result from the following four lemmata.

Lemma 1: A process that precedes all others can advance at least one stage.

Proof. Let p_i find, when it starts line 6, that

$$j = q[i] > q[k],\ \forall k \neq i, \tag{1}$$

then the condition in lines 6–7 is not satisfied and p_i can advance to stage $j+1$. Since checking the condition is not a primitive, (1) may become untrue during the check: some p_r may set $q[r] = j$; several could do so, but as soon as the first of them also modifies $turn[j]$, p_i can proceed (assuming it tries; this assumption will dog us throughout the proof – but will be kept tacit from now on). Moreover, once more than one additional process joins stage j, p_i can be overtaken.

Lemma 2: When a process passes from stage j to stage $j+1$, *exactly* one of the following claims holds

(a) it precedes all other processes, or

(b) it is not alone at stage j.

Remark: "process p_i advances a stage" is not a globally available fact. We actually refer to the instant of storing a new value in $q[i]$.

Proof: Let p_i be about to advance $q[i]$. Then either condition (1) holds – and then line (a) holds as well, or $turn[j] \neq i$, with the implication that p_i is not the last, among all processes currently in lines 3–10 of their programs, to enter stage j. Regardless of how many processes modified $turn[j]$ since p_i did, there is a last one, p_r, for which the condition in his line 7 is true. This process makes line (b) in the Lemma hold. Note that it is possible that p_i proceeds to modify $q[i]$ on the strength of its finding that condition (1) is true, and in the meantimes another process destroys this condition, thereby establishing possibility (b).

Lemma 3: If there are (at least) two processes at stage j, there is (at least) one process at each stage k, $1 \leq k \leq j-1$.

Proof: By induction on j. The claim is void for $j=1$. For $j=2$, we use Lemma 2: when there is a process at stage 2 another one (or more) will join it only when the joiner leaves behind a process in stage 1. That one, so long as it is alone there cannot advance, again by Lemma 2.

Assume the Lemma holds for stage $j-1$: if there are two (or more) at stage j, consider the instant the last of them joined in. At that time, there were (at least) two at stage $j-1$ (Lemma 2), and by the induction assumption, all preceding stages were occupied. By Lemma 2 none of these stages could have vacated since.

Lemma 4: The maximal number of processes that can be at stage j is $n-j+1$, $1 \le j \le n-1$.

Proof. By arithmetic, from Lemma 3: if stages 1 through $j-1$ contain at least one process, there are $n-(j-1)$ left at most for stage j. If any of those stages is "empty", Lemma 3 implies there is at most one process at stage j.

Hence stage $n-1$ contains at most two processes. If there is only one there, and another is at its CS, Lemma 2 says it cannot advance to enter enter its CS. When stage $n-1$ does contain two, there is no process left to be at stage n (CS), and one of the two may enter its CS. For the one remaining process the condition in its line 7 holds. It would be nice to claim that since the CS is given the stage-index n, ME follows directly from the arithmetic of the last Lemma; but strictly speaking, that was proved using double occupancy, and to extend it to stage n would require exactly the above argument.

We have shown that ME holds. The other claims follow immediately: clearly there is no deadlock -- there is one process that precedes all others or is with company at the highest occupied stage, which it was not the last to enter, and for such a process the condition of its line 7 does not hold.

Similarly, if a process tries continually to advance, no other process can pass it; at worst, it entered stage 1 when all others were in their entry protocols; they may all enter stage n before it does – but no more. Hence the claimed "linear" waiting.

So far "if it were continually trying"; on a machine with less than n processors an adverse scheduler may detain a process for a longer time, but whenever it is scheduled and it is at a stage which it was not the last to enter, it can advance a stage. Using careful (and contrary) scheduling it can be made to advance one stage only, with unlimited number of visits by the other processes to their critical sections having taken place. Hence the caveat at the statement of this property.

4. DISCUSSION

As mentioned by Peterson, this is neither the strongest nor cheapest protocol, in that others that satisfy further properties or use fewer variables exist (Raynal (1986) is a good source). It is, however, a very nicely structured one, and shows clearly the relation to the 2–process protocol from which it was fashioned.

As the proof unfolds, alert students will find ample opportunity to question various statements and raise numerous issues that relate to atomicity, scheduling, properties of the supporting hardware and more. It is precisely for this reason that I found proving the protocol a useful teaching device.

REFERENCES

[1] G.L. Peterson, Myths About the Mutual Exclusion Problem, Inf. Proc. Lett. **12** #3, 115–116 (1981).

[2] M. Raynal, *Algorithms for Mutual Exclusion,* North Oxford Academic Pub. (1986).