A Note on Lamport's Mutual Exclusion Algorithm

Tai-Kuo Woo

Department of Computer and Information Sciences
Jacksonville University
Jacksonville, FL 32211

Abstract

The Lamport's mutual exclusion exclusion algorithm, also known as the bakery algorithm, is very simple and easy to implement. However, the ticket numbers assigned to the processes wishing to enter the critical section can become indeterminately large. This paper resolves this problem by allowing the ticket numbers to move around a ring of length $2N$, where N is the number of processes. The comparison of two ticket numbers is based the distance between them and their magnitudes.

1 Statement of the Problem

The Lamport's mutual exclusion algorithm [1], which is based on a scheduling algorithm commonly used in bakeries, is one of many N-process software solutions. It is simple and elegent. What it does and why it works are completely straightforward. A process wishing to enter the critical section is assigned a ticket number. The process with the smallest ticket number has the highest precedence for entering the critical section. In case more than one process receives the same ticket number, the process with the smallest process number enters the critical section. When a process exits the critical section, it resets its ticket number to zero. The ticket number assigned is calculated by adding one to the largest of the ticket numbers currently held by the processes wishing to enter the critical section and the process already in the critical section. The procedure below shows the protocol for process i wishing to enter the critical section.

The shared data structures include:

- *choosing*: an array $[0..N-1]$ of boolean. The elements are initialized to false.
- *num*: an array $[0..N-1]$ of integer. The elements are initialized to zero.

```
repeat
1  choosing[i] := true;
2  num[i] := MAX(num[0], num[1], num[2], ..., num[N − 1]) + 1;
3  choosing[i] := false;
4    for j := 0 to N − 1 do
5      begin
6        while choosing[j] do;
7        while num[j] ≠ 0  ∧ (num[j], j) < (num[i], i) do;
8      end;
9    critical section
```

10 $num[i] := 0$;
11 *remainder section*
until $false$;

The function MAX returns the largest value in the array num, and the logical expression $(num[j], j) < (num[i], i)$ is defined as $(num[j] < num[i]) \vee ((num[i] = num[j]) \wedge (j < i))$.

If there is always at least one process holding a ticket number while the maximum is being computed, the ticket number becomes indeterminately large. For $N > 2$, no simple solutions are available.

2 The Solution

The solution we propose is to let the ticket numbers move around a ring of length $2N$, i.e., ranging from 0 to $2N - 1$. To determine which of any two ticket numbers has precedence for entering the critical section, we compute the difference of these two ticket numbers. If the difference is greater than or equal to N, the process with the larger ticket number has precedence over the process with the smaller ticket number. If the difference is less than N, the process with the smaller ticket number has precedence over the process with larger ticket number. For instance, a process with ticket number 8 has precedence over a process with ticket number 10 for N equal 6, since the difference of these two numbers is less than than N. However, a process with ticket number 10 has precedence over a process with ticket number 2, since the difference of these two numbers is greater than N. As a result, to test if the process with ticket number A has precedence over the process with ticket number B, the logical expression that should be used is $(|A - B| \geq N) \oplus (A < B)$, where $\oplus$ is the logical operator exclusive or, instead of $(A < B)$.

The procedure for solving the problem of the unbounded number in the bakery algorithm now becomes the following:

repeat
1 $choosing[i] := true$;
2 $num[i] := (MODMAX(num[0], num[1], num[2], ..., num[N-1]) + 1)\ MOD\ 2N$;
3 $choosing[i] := false$;
4 for $j := 0$ *to* $N - 1$ do
5 begin
6 while $choosing[j]$ do;
7 while $num[j] \neq -1\ \wedge (num[j], j) < (num[i], i)$ do;
8 *end*;
9 *critical section*
10 $num[i] := -1$;
11 *remainder section*
until $false$;

In the procedure, the logical expression $(num[j], j) < (num[i], i)$ is defined as $((|num[j] - num[i]| \geq N) \oplus (num[j] < num[i])) \vee ((num[i] = num[j]) \wedge (j < i))$.

The function $MODMAX$ is defined as follows:

```
Function MODMAX(num : integer) : integer;
begin
  while num[l] = −1 do l := l + 1;  /* l is initialized to 0 */
  max := num[l];  /* Establish temporary maximum */
  for k := l + 1 to N − 1 do
    If ((|max − num[k]| ≥ N) ⊕ (max < num[k])) ∧ (num[k] > −1)
      then max := num[k];
  return max;
end;
```

What the function $MODMAX$ does is check each element of array num. If the element $num[k]$ equals -1, process k does not want to enter the critical section. Otherwise, if element $num[k]$ has lower precedence than the temporary max for entering the critical section in the sense defined in the beginning of this section, we update the temporary max. The function returns the ticket number which has the lowest precedence for entering the critical section.

3 Conclusion

The approach we have taken to resolve the unbounded number problem requires two major changes. First, instead of computing the maximum of the ticket numbers, we compute $MODMAX$. As indicated in the function $MODMAX$, the additional computations required include evaluating one logical expression and an extra procedure call. After $MODMAX$ is computed, an additional modular operation needs to be applied to the expression $MODMAX + 1$. Second, the logical expression $(num[j], j) < (num[i], i)$ needs to be evaluated in a slightly different way. In the worst case, an additional N evaluations of the expression $|num[j] - num[i]| \geq N$ are required.

References

[1] L. Lamport. A New Solution of Dijkstra's Concurrent Programming Problem. Communication of the ACM, Vol. 17, No. 8, 1974, pp. 453-455.