Bakery Algorithm

Introduction

This algorithm solves the critical section problem for $n$ processes in software. The basic idea is that of a bakery; customers take numbers, and whoever has the lowest number gets service next. Here, of course, “service” means entry to the critical section.

Algorithm

```
var choosing: shared array [0..n-1] of boolean;

number: shared array [0..n-1] of integer;
...

repeat
  choosing[i] := true;
  number[i] := max(number[0], number[1], ..., number[n-1]) + 1;
  choosing[i] := false;
  for j := 0 to n-1 do begin
    while choosing[j] do (* nothing *);
    while number[j] <> 0 and
      (number[j], j) < (number[i], i) do
      (* nothing *);
  end;
  (* critical section *)
  number[i] := 0;
  (* remainder section *)
until false;
```

Comments

lines 1-2: Here, `choosing[i]` is true if $P_i$ is choosing a number. The number that $P_i$ will use to enter the critical section is in `number[i]`; it is 0 if $P_i$ is not trying to enter its critical section.

lines 4-6: These three lines first indicate that the process is choosing a number (line 4), then try to assign a unique number to the process $P_i$ (line 5); however, that does not always happen. Afterwards, $P_i$ indicates it is done (line 6).

lines 7-12: Now we select which process goes into the critical section. $P_i$ waits until it has the lowest number of all the processes waiting to enter the critical section. If two processes have the same number, the one with the smaller name – the value of the subscript – goes in; the notation “$(a,b) < (c,d)$” means true if $a < c$ or if both $a = c$ and $b < d$ (lines 9-10). Note that if a process is not trying to enter the critical section, its number is 0. Also, if a process is choosing a number when $P_i$ tries to look at it, $P_i$ waits until it has done so before looking (line 8).

line 14: Now $P_i$ is no longer interested in entering its critical section, so it sets `number[i]` to 0.
Bogus Bakery Algorithm

Introduction

Why does Lamport's Bakery algorithm use a variable called choosing (see the algorithm, lines 1, 4, 6, and 8)? It is very instructive to see what happens if you leave it out. This gives an example of mutual exclusion being violated if you don't put choosing in. But first, the algorithm (with the lines involving choosing commented out) so you can see what the modification was:

```
Algorithm
1 var (*choosing: shared array [0..n-1] of boolean; *)
2 number: shared array [0..n-1] of integer;
3 ...
4 (* choosing[i] := true; *)
5 number[i] := max(number[0],number[1],...,number[n-1]) + 1;
6 (* choosing[i] := false; *)
7 for j := 0 to n-1 do begin
8 (* while choosing[j] do *)
9 while number[j] <> 0 and
10 (number[j], j) < (number[i],i) do
11 (* nothing *);
12 end;
13 (* critical section *)
14 number[i] := 0;
15 (* remainder section *)
16 until false;
```

Proof of Violation of Mutual Exclusion

Suppose we have two processes just beginning; call them p₀ and p₁. Both reach line 5 at the same time. Now, we'll assume both read number[0] and number[1] before either addition takes place. Let p₁ complete the line, assigning 1 to number[1], but p₀ block before the assignment. Then p₁ gets through the while loop at lines 9-11 and enters the critical section. While in the critical section, it blocks; p₀ unblocks, and assigns 1 to number[0] at line 5. It proceeds to the while loop at lines 9-11. When it goes through that loop for j = 1, the condition on line 9 is true. Further, the condition on line 10 is false, so p₀ enters the critical section. Now p₀ and p₁ are both in the critical section, violating mutual exclusion.

The reason for choosing is to prevent the while loop in lines 9-11 from being entered when process j is setting its number[j]. Note that if the loop is entered and then process j reaches line 5, one of two situations arises. Either number[j] has the value 0 when the first test is executed, in which case process i moves on to the next process, or number[j] has a non-zero value, in which case at some point number[j] will be greater than number[i] (since process i finished executing statement 5 before process j began). Either way, process i will enter the critical section before process j, and when process j reaches the while loop, it will loop at least until process i leaves the critical section.