Test and Set Solution

Introduction

This algorithm solves the critical section problem for *n* processes using a Test and Set instruction (called TaS here). This instruction does the following function atomically:

```
function TaS(var Lock: boolean): boolean;
begin
      TaS := Lock;
      Lock := true;
end;
```

Algorithm

```
 1   var          waiting: shared array [0..n-1] of boolean;
 2                Lock: shared boolean;
 3                j: 0..n-1;
 4                key: boolean;
           ...
 5           repeat                    (* process Pi *)
 6                waiting[i] := true;
 7                key := true;
 8                while waiting[i] and key do
 9                     key := TaS(Lock);
10                waiting[i] := false;
11                (* critical section goes here *)
12                j := i + 1 mod n;
13                while (j <> i) and not waiting[j] do
14                     j := j + 1 mod n;
15                if j = i then
16                     Lock := false
17                else
18                     waiting[j] := false;
19           until false;
```

Comments

lines 1-2: These are global to all processes, and are all initialized to `false`.

lines 3-4: These are local to each process P_i and are uninitialized.

lines 5-10: This is the entry section. Basically, *`waiting[i]`* is `true` as long as P_i is trying to get into its critical section; if any other process is in that section, then *`Lock`* will also be true, and P_i will loop in lines 8-9. Once P_i can go on, it is no longer waiting for permission to enter, and sets *`waiting[i]`* to `false` (line 10); it then proceeds into the critical section. Note that *`Lock`* is set to `true` by the *`TaS`* instruction in line 9 that returns `false`.

lines 12-18: This is the exit section. When P_i leaves the critical section, it must choose which other waiting process may enter next. It starts with the process with the next higher index (line 12). It checks each process to see if that process is waiting for access (lines 13-14); if no-one is, it simply releases the lock (by setting *`Lock`* to `false`; lines 15-16). However, if some other process P_j is waiting for entry, P_i simply shanges *`waiting[j]`* to `false` to allow P_j to enter the critical section (lines 17-18).