Test and Set Solution

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;
```

The solution is:

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

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

lines 3–4: These are local to each process `i` and are uninitialized.

lines 6–11: This is the entry section. Basically, `waiting[i]` is **`true`** as long as process `i` is trying to get into its critical section; if any other process is in that section, then `Lock` will also be **`true`**, and process `i` will loop in lines 9–10. Once process `i` can go on, it is no longer waiting for permission to enter, and sets `waiting[i]` to **`false`** (line 11); 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 13–19: This is the exit section. When process `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 13). It checks each process to see if that process is waiting for access (lines 14–15); if none is, it simply releases the lock (by setting `Lock` to **`false`**; lines 16–17). However, if some other process process `j` is waiting for entry, process `i` simply changes `waiting[j]` to **`false`** to allow process `j` to enter the critical section (lines 18–19).