Shared Memory Algorithms for Mutual Exclusion

The complexity of the solution depends on the grain of atomicity.

Atomic reads and writes
Using read-modify-writes
DEC’s LL and SC primitives

Solutions using atomic reads and writes

Intellectually challenging, and has a long history. The real world does not care for it very much. To realize the challenge, try to write your own algorithms first, and prove that it satisfies all properties. It is not easy!
Warm Up

{process 0} {process 1}

define f : shared Boolean (initially false)

do true ‡
   do f = true ‡ skip od;
   f:=true;
   CS;
   f:= false;
od

do true ‡
   do f =true ‡ skip od;
   f:=1;
   CS;
   f:=false
od

Why doesn’t it work?

Historic Dekker’s solution (see homework 3)
**Peterson’s two-process solution**

```
program  peterson;
define  flag[0], flag[1]  : shared boolean;
        turn: shared integer
initially  flag[0] = false, flag[1] = false, turn = 0 or 1

{program for process 0}
do  true □
1:  flag[0] = true;
2:  turn = 0;
3:  do (flag[1] □ turn =0) □ skip od
4:  critical section;
5:  flag[0] = false;
6:  non-critical section codes;
od

{program for process 1}
do  true □
7:  flag[1] = true;
8:  turn = 1;
9:  do (flag[0] □ turn = 1) □ skip od;
10: critical section;
11:  flag[1] = false;
12:  non-critical section codes;
od
```
PROOFS

1. At most one process enters its CS.
   Let 0 be in CS. Can 1 enter its CS?
   0 in CS \( \implies \) flag[0] = false OR turn = 1 OR both.
   To enter CS, 1 must see flag[0] = false OR turn = 0 OR both.
   But 0 in CS \( \implies \) flag[0] = true! So turn = 0 should hold.

Case 1.
   process 0 reads \textbf{flag[1]} = false in step 3
     \(\Box\) process 1 has not executed step 7
     \(\Box\) process 1 eventually sets \textbf{turn} to 1 (step 8)
     \(\Box\) process 1 checks \textbf{turn} (step 9) and finds \textbf{turn} = 1
     \(\Box\) process 1 waits in step 9 and cannot enter its CS

Case 2.
   process 0 reads \textbf{turn} = 1 in step 3
     \(\Box\) process 1 executed step 8 after 0 executed step 2
     \(\Box\) in step 9 process 1 reads \textbf{flag[0]} = true and \textbf{turn} = 1
     \(\Box\) process 1 waits in step 9 and cannot enter its CS
2. *Deadlock is impossible.*

Hint: $(\neg flag[1] \land turn = 0) \land (\neg flag[0] \land turn = 1) = \text{false}$

3. *Progress (eventual entry into CS)*

Study this yourself.
Solution using read-modify-write

![Diagram of shared memory and processes]

\[ TS (r, x) \equiv <r := x; x := 1> \{\text{atomic operation}\} \]

program TS (for any process);
define x: shared integer;
r: integer (private);
initially x=0;

do true []
  do r \neq 0 [] TS(r,x) od;
  critical section;
x := 0
od
**Solution using LL and SC primitives**

LL(r, x) is like a machine instruction load (i.e. \( r := x \)). In addition, the address x is automatically recorded by the system.

SC(x, r) is like a machine instruction store (i.e. \( x := r \)). However, if the process executing SC is the first process to do so after the last LL, then the store operation “succeeds,” which is reported by returning a value 1 into r. Otherwise the store operation “fails,” the value of x remains unchanged, and a 0 is returned into r.

**Example. Mutual Exclusion using LL and SC**

```
program  mutex (for any process);
define    x: shared integer, r: integer (private);
initially x=0;
try:      LL(r, x);
  if r ≠ 0 \fi goto try fi;
  r:= 1
  SC(x, r)
  if r = 0 \fi goto try fi
  critical section;
  x:=0;
non-critical section codes;
od
```