

Correction algorithm de Kessel: Initialement : $b=\{false, false\}$, $turn=quelconque$

On suppose que T0 est en SC. On a deux cas

Cas 1: T0 a lu $b[1]=false$

Thread 0

```
b[0]=true;  
local[0]=turn[1];  
turn[0]=local[0];  
while ((b[1]==true && (local[0]==turn[1])))
```

critical section

$b[0] = false;$

Thread 1

```
b[1]=true;  
local[1]=1-turn[0];  
turn[1]=local[1];  
while ((b[0]==true && (local[1]!=turn[0])))
```

critical section

$b[1] = false;$

Dans ce cas T1 commence à exécuter le protocole d'entrée après que T0 soit entré donc on a la séquence

T0 en SC $\rightarrow b[1]=true \rightarrow local[1]=1-turn[0] \rightarrow turn[1]=local[1] \rightarrow while...$

T1 va voir $b[0]==true$ car $b[0]=true \rightarrow b[1]==true \rightarrow b[0]==true$

Et T1 va voir $local[1] \neq turn[0]$ car après que T1 écrive $local[1]=1-turn[0]$ ni la valeur de $local[1]$ n'est modifiée ni la valeur de $turn[0]$ par T0 (car l'écriture par T0 $turn[0]=local[0] \rightarrow local[1]=1-turn[0]$ grâce à l'hypothèse \rightarrow (notée par la flèche rouge)

Cas 2: T0 a lu `local[0] != turn[1]`

Dans ce cas la valeur de `turn[1]` a dû changer entre le moment où T0 exécute `local[0]=turn[1]` et le moment où il teste `local[0]==turn[1]`. Donc T1 a écrit `turn[1]=local[1]` avant le test par T0.

Thread 0

```
b[0]=true;
local[0]=turn[1];
turn[0]=local[0];
while ((b[1]==true && (local[0]==turn[1])))
```

critical section

b[0] = false;

Thread 1

```
b[1]=true;
local[1]=1-turn[0];
turn[1]=local[1];
while ((b[0]==true && (local[1]!=turn[0])))
```

critical section

b[1] = false;

Une fois que T0 est entré en SC les variables ne sont plus modifiées, on a toujours

(1) `local[0] != turn[1]` et `turn[1] == local[1]`

De même on a toujours `turn[0] == local[0]` car les écritures de `local[0]` et `turn[0]` → l'entrée en SC par T0. On a finalement `turn[0]=local[0] != turn[1]=local[1]` donc `local[1] != turn[0]` et la deuxième condition d'attente est satisfaite pour T1.

Comme `local[0]=turn[1]` par T0 → `turn[1]=local[1]` → `b[0]==true` par T1, T1 voit `b[1]==true` et la première condition est satisfaite. T1 attend donc.

Les deux threads ne peuvent pas être interbloqués. On devrait avoir que T0 et T1 exécute la boucle **while** sans fin, c'est-à-dire
`local[0]==turn[1]` et `local[1]!=turn[0]`

On a toujours `turn[0]=local[0]` et `turn[1]=local[1]`. On devrait donc avoir

`turn[0]==local[0]==turn[1]` et `turn[1]==local[1]!=turn[0]`

Ce qui est impossible.

Le protocole est starvation-free.

Si T1 est en attente dans la boucle car T0 est en SC.

T0 sort de la section critique et écrit **b[0]=false**.

Si on suppose que T1 ne voit pas l'écriture et T0 recommence le protocole d'entrée avant le test par T1.

T0 arrivera donc à la boucle **while**.

Il verra `b[1]==true` car T1 exécute la boucle **while** et

`b[1]=true` → **while** par T1

T0 verra `local[0]==turn[1]` car la valeur de `turn[1]` n'a pas changée entre le moment où T0 écrit `local[0]=turn[1]` et le test.

On a vu au point précédent que T1 doit alors entrer en SC.