

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

Cas 1: Impossible car il y a un cycle

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

$b[1] = false;$

Cas 2:

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 lit $b[0]==false$ alors il rentre en SC. T0 va lire $b[1]==true$ et la seconde condition $local[0]==turn[1]$ est vérifiée car T0 a écrit $local[0]=turn[1]$ et la valeur de $turn[1]$ n'a pas été modifiée après.

Cas 3:

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;

Dans ce cas c'est T0 qui entre en SC et T1 qui reste bloqué par le même argument qu'au Cas 2.

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;

On a maintenant dans tous les cas $b[0]==true$ et $b[1]==true$ dans les boucles while. Il faut maintenant discuter les conditions $local[0]==turn[1]$ et $local[1]!=turn[0]$

Cas 4.1: Impossible

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;

Cas 4.2

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**;

On a

local[0]=turn[1];

turn[0]=local[0];

while ((b[1]==true && (local[0]==turn[1]))) T0 est bloqué tant que la prochaine instruction n'est pas exécutée

turn[1]=local[1];

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

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**;

local[1]=1-turn[0];

Supposons que T1 entre en SC, c'est-à-dire qu'on a local[1]==turn[0]

On sait que turn[1]==local[1], donc turn[1]==local[1]==turn[0] et aussi turn[0]==local[0] donc T0 reste bloqué

Supposons que T1 est bloqué, c'est-à-dire qu'on a local[1]!=turn[0]

On sait que turn[1]==local[1], donc turn[1]==local[1]!=turn[0] et aussi turn[0]==local[0] donc T0 est maintenant débloqué et va entrer en SC.

Cas 4.3

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;
```

On a

```
local[0]=turn[1];
```

T1 est bloqué tant que la prochaine instruction n'est pas exécutée

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;
```

```
local[1]=1-turn[0];
```

```
turn[1]=local[1];
```

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

```
turn[0]=local[0];
```

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

Supposons que T0 entre en SC, c'est-à-dire qu'on a $local[0] \neq turn[1]$

On sait que $turn[0] == local[0]$, donc $turn[0] == local[0] \neq turn[1]$ et aussi $turn[1] == local[1]$ donc T1 reste bloqué

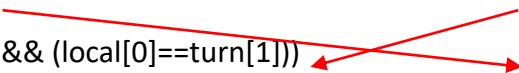
Supposons que T0 est bloqué, c'est-à-dire qu'on a $local[0] == turn[1]$

On sait que $turn[0] == local[0]$, donc $turn[0] == local[0] == turn[1]$ et aussi $turn[1] == local[1]$ donc T1 est maintenant débloqué et va entrer en SC.

Cas 4.4

Thread 0

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



critical section

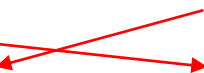
b[0] = **false**;

Pour le test de T0 on a local[0]==turn[0]== turn[1]

Pour le test de T1 on a turn[1]==local[1]!=turn[0]

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**;

On voit que les deux conditions sont exclusives, l'une est vrai et pas l'autre, il y a donc un et une seul thread qui entre en SC.