THE DEKKER'S ALGORITHM

Two processes use a respective variable to indicate that they want to enter a critical section. However using just that could lead to a deadlock. So they use a tie-breaker "turn" to indicate whose turn it is to wait. So, each process says it wants to enter CS but also that it is its turn to wait. In the end, a process only waits if the other process wants to enter CS as well as it is its own turn to wait. This tie-breaker prevents deadlock.

CODE: https://repl.it/@wolfram77/dekkers-algorithm#Main.java

```
import java.util.concurrent.*;


class Main {
static int c1, c2;
static int turn;


static void process1() {
new Thread(() -> {
try {
while(true) {
c1 = 1;
turn = 1;
log("1: waiting");
while(c2==1 && turn==1) Thread.sleep(10);
log("1: enter critical section");
Thread.sleep((long)(Math.random()*1000));
log("1: exits critical section");
c1 = 0;
}
}
catch(InterruptedException e) {}
}).start();
}


static void process2() {
new Thread(() -> {
try {
while(true) {
c2 = 1;
```

```
turn = 2;
log("2: waiting");
while(c1==1 && turn==2) Thread.sleep(10);
log("2: enter critical section");
Thread.sleep((long)(Math.random()*1000));
log("2: exits critical section");
c2 = 0;
}
}
catch(InterruptedException e) {}
}).start();
}


public static void main(String[] args) {
process1();
process2();
}
static void log(String x) {
System.out.println(x);
}
}
```

OUTPUT: https://dekkers-algorithm.wolfram77.repl.run

```
1: waiting
1: enter critical section
2: waiting
1: exits critical section
1: waiting
2: enter critical section
2: exits critical section
2: waiting
1: enter critical section
1: exits critical section
1: waiting
2: enter critical section
2: exits critical section
2: waiting
1: enter critical section
1: exits critical section
1: waiting
2: enter critical section
...
```