Assignment 16.

State Dekker's algorithm. List its properties and prove each of them.

For process 0:

```
inter[0] = True
while (inter[1] == True) {
    if (turn == 1) {
        inter[0] = False
        while (turn == 1) {}
        inter[0] = True
    }
}
<CS>
turn = 1
inter[0] = False
```

For process 1:

```
inter[1] = True
while (inter[0] == True) {
        if (turn == 0) {
            inter[1] = False
            while (turn == 0) {}
            inter[1] = True
        }
}
<CS>
turn = 0
inter[1] = False
```

Mutual exclusion:

Assume for the sake of contradiction that Dekker’s algorithm do not provide mutual exclusion. The only case for this to be true is that both Boolean inter[0] and inter[1] are **false** at the same time (while both processes 0 and 1 are in the CS).
Assuming first that inter[1] is false. Then, obviously inter[0] must be true cause there is only two ways to state inter[0] as false: the first one is into the busy waiting, but recall that inter[1] false, we must have “turn == 0” since only process 1 may state inter[1] = false and may state turn = 0 before of already have turn == 0. Thus, if turn = 0, process 0 can’t enter in the if statement (turn == 1) and then state the inter[0] as false. The second way to state inter[0] as false is after the CS but before to have a chance to reenter once again, process 0 must state turn to 1 and then either process 1 is interested and is blocked in the while (turn == 0) statement and then at the time he will get running time he will state inter[1] as true, or process 1 is not interested and then if he gets interested he also must set inter[1] as false.
The proof when assuming inter[0] is false is the same than above.
Then in all the case there is no possibilities for inter[0] and inter[1] to be false at the same time, contradicting our assumption, and proving the claim.

■

Deadlock free:

Assume by contradiction that Dekker's algorithm is not deadlock free. Then both process 0 and 1 are stuck forever in the entry of the CS. Trivially turn can't be at the same time equal to 0 and 1, so both process can't be stuck in the while (turn == 0/1) loop. Thus, by the same way, even if both value of turn[0] and turn [1] are true (if not there is not process blocked and one of them can enter in the CS) since turn must be either 1 or 2, when the one which can over the if (turn = 0/1) gets running time, he will state inter[0/1] as false and then wait in the while loop until the second process will finish his work in the CS. And then, when the second process will have running time, he will enter in the CS, contradicting that both processes are stuck forever in the entry code.

■

Starvation free:

To proving that the Dekker's algorithm provide starvation freedom, we need to show that there is no process (either o or 1) which is stuck forever in the entry code. We're doing thing by using the meaning of the variable turn. Assuming turn = 0 at the beginning, and both processes get interested (cause if not the one which is interested should not have any problem to enter in the CS). Having turn = 0 we have inter[1] = false. While the process 1 is block in the while (turn == 0) when the process gets running time he enters in the CS, when he is out, he states the turn value to 1. And this ensure that even if process 1 is re-interested by reenter in the CS, since the value of turn is 1, he will not enter in the CS before process 1. This provide a starvation freedom, because we proved that there is no process which is blocked forever in the entry code.

■