

# Assignment #1

Multiprocessor Synchronization Algorithms (20225401)

**Assignment solutions should be prepared by a text editor and not hand-written. They should be sent via e-mail (in Word or pdf format) to [rotemra@cs.bgu.ac.il](mailto:rotemra@cs.bgu.ac.il) by midnight, Sunday, December 4<sup>th</sup>.** This assignment must be solved **individually and not in teams.**

## Mutual exclusion

1. Modify Peterson's 2-process algorithm, without using additional shared-memory registers, so that its contention-free complexity (i.e., its complexity when a process runs alone) is only *three* accesses: two accesses in the entry code and one in the exit code. Prove that your algorithm maintains mutual exclusion and starvation freedom.
2. Prove in a formal and accurate manner that the one-bit algorithm, which we have seen in class, satisfies mutual exclusion.
3. Consider the code of Lamport's fast mutual exclusion algorithm shown in the presentation.
  - a. Assume we remove line 12 (*await slow-lock=0*). Does the algorithm still satisfy both mutual exclusion and deadlock-freedom? Either justify shortly why it does or provide an execution showing otherwise.
  - b. Assume we remove both lines 12 and 5 (*await slow-lock=0*). Does the algorithm still satisfy mutual exclusion and deadlock-freedom? Either justify shortly why it does or provide an execution showing otherwise.
4. The following pseudo-code implements a mutual exclusion algorithm for 2 processes with IDs 1 and 2. The algorithm uses 2 registers: register *x* can assume values 0, 1 or 2, and register *y* can assume values 0 or 1.

Code for process  $i \in \{1,2\}$

Initially  $x=0$  and  $y=0$ .

```
1 start: x:=i
2 if y ≠ 0
3   await y=0
4   goto start
5 y:=1
6 if x ≠ i
7   y:=0
8   await x=0
9   goto start
10 critical section
11 y:=0
12 x:=0
```

- a. Does the algorithm satisfy mutual exclusion? Either prove, or describe an execution that shows it does not.
- b. Does the algorithm satisfy deadlock-freedom? Either prove, or describe an execution that shows it does not.
- c. Does the algorithm satisfy starvation-freedom? Either prove, or describe an execution that shows it does not.
- d. Assume the algorithm is used by **3 processes**, with IDs 1, 2, 3 (that is, the code above is for  $i \in \{1,2,3\}$ ). Will the algorithm satisfy mutual exclusion in this case? Either prove or describe an execution that shows it does not.

## Leader Election

The following question deals with leader election in rings. As always, we assume that the ring is **oriented**, i.e. processes have a consistent view of left and right

5. Recall the  $\Omega(n \log n)$  lower bound that we have seen on the number of messages required for electing a leader in an asynchronous ring. The proof of the lower bound relies on two additional assumptions: (a) the process with the maximum identifier is elected, and (b) all processes need to learn the identifier of the elected leader.

Prove that the lower bound holds also when these requirements are removed.

**בהצלחה!**