Third Homework of Concurrent Systems

Exercise 1. Let P_1, P_2 and P_3 be three processes each running an operation op_i. At every moment, we want the number of op_i that are terminated or in execution (not the ones that are suspended, waiting to be executed) to be at most one more than the number of op_{i+1}. Write a solution using semaphores, and remember to specify the domain of each semaphore you use and its initial value.

Exercise 2. Consider the following solution for the dining philosophers:

```
Array [1..5] of semaphores: fork = [1, 1, 1, 1, 1];

Philosopher i:
Repeat forever
    think;
    if (i mod 2 = 0)
       then fork[i].down();fork[(i+1) mod 5].down();
       else fork[(i+1) mod 5].down();fork[i].down();
    eat;
    fork[(i+1) mod 5].up(); fork[i].up();
```

Prove that it is deadlock free.

Exercise 3. Consider the following simplification for the multiple producers/consumers problem:

```
produce(v) ::=                  consume() ::=
    FREE.down();                    BUSY.down();
    SP.down();                      SC.down();
    i ← IN;                         i ← OUT;
    IN ← (IN+1) mod k;              OUT ← (OUT+1) mod k;
    SP.up();                        SC.up();
    BUFF[i] ← v;                    r ← BUFF[i];
    BUSY.up();                      FREE.up();
    return();                       return(r);
```

where semaphores FREE/BUSY/SP/SC have the same meaning and are initialized in the same way as in section 3.2.2 (algorithm in Fig. 3.7). Is this simplified algorithm correct? To show that it is correct, a proof has to be given. To show that it is incorrect, a counter-example has to be exhibited.

Exercise 4. Program a monitor for three processes that have the following behavior: processes A and B may execute operations op_A and op_B (in any order and as many times as they want) only after process C has completed at least one invocation of operation op_C.