DAT 103
Datamaskiner og operativsystemer (Computers and Operating Systems)

Supplementary exercises (Set 5)

Problem 1

Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

Solution.
The condition that would break a process out of the spinlock can be obtained only by executing a different process. If the process that is stuck the spinlock does not give away the processor, no other processes can set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and therefore can modify the program state in order to release the first process from the spinlock.

Problem 2

Describe how the `signal()` operation associated with monitors differs from the corresponding operation defined for semaphores.

Solution.
The `signal()` operation associated with monitors is not persistent in the following sense: if a signal is performed and if there are no wait- ing threads, then the signal is simply ignored, and the system does not remember that the signal took place. If a subsequent wait oper- ation is performed, then the corresponding thread simply blocks. In semaphores, every signal results in a corresponding increment of the semaphore value even if there are no waiting threads. A future wait operation would immediately succeed because of the earlier increment.

Problem 3

Describe how deadlock is possible with the dining-philosophers problem.

Solution.
If all philosophers simultaneously pick up their left forks, all their right forks will become unavailable, and will block forever to wait for the right forks to become available, thus, a deadlock occurs.

Problem 4

Let `l1` and `l2` be locks, and `x` a shared `int` variable initialised to `0`. Consider two processes P_1 and P_2 that concurrently run the following pieces of code:

```
1 //  Run by P1  //
2 acquire(l1);
3 x++;
4 release(l1);
```

```
5 //  Run by P2  //
6 acquire(l2);
7 x--;
8 release(l2);
```

List all possible values that `x` could have when both processes have finished.

Solution.
0, 1, -1

Problem 5

Now, assume P_1 and P_2 concurrently run the two methods in Listing 1 to access their critical sections instead. Complete Listing 1 by filling in the boolean condition of the while-loops in Lines 8 and 18 such that it is a solution to the critical section problem, i.e., it guarantees both mutual exclusion and progress.

```
1  bool lock1 = false; bool lock2 = false;
2  int i;
3
4  //  Run by P1  //
5  while (true) {
6    lock1 = true;
7    i = 1;
8    while (???  // Fill in your code //) {skip} ;
9    ;; critical section
10   lock1 = false ;
11   ;; non critical section
12 }
13
14 //  Run by P2  //
15 while (true) {
16   lock2 = true;
17   i = 2;
18   while (???  // Fill in your code //) {skip} ;
19   ;; critical section
20   lock2 = false ;
21   ;; non critical section
22 }
```

Listing 1:

Solution.

Line 8: `lock2 && i==1`
Line 18: `lock1 && i==2`