Series 01— 20.09.2017 – v1.0

Introduction to Concurrency

Exercise 1 (7 Points)

1. Do recent central processing units (CPUs) of desktop PCs support concurrency? Why became concurrency

for many software applications very important these days?

Concurrency is supported by single core (single thread) CPUs, e.g. by time-sharing. With the increasing number of cores and thus threads in modern CPU's there is a huge advantage for concurrency because threads/processes can e.g. run parallel. So recent desktop CPU's support concurrency.

As in 'Concepts and Notations for Concurrent Programming' (http://scgresources.unibe.ch/Literature/CP/Andr83aSurvey.pdf), "...,concurrent programming no longer is the sole province of those who design and implement operating systems". It has been important for a long time. But the CPU evolution (price, cores, multisockets, ...) is one of the main reason it is getting more important, since we can now benefit more of it in software applications beside operating systems. Some examples are, (as in lecture notes1: p15/15.5:) ) : - Reactive programming - Real-time programming - Simulation - Parallelism - Distribution, which where becoming increasingly important.

2. What is safety? Give one concrete example of a safety violation.

Safety = ensuring consistency (slides p.18)

'One way of thinking about safety is that the data of our program are not corrupted.

Another way of thinking about safety is that the program does not reach a “bad state”. The “bad state” can manifest itself as inconsistent memory, or perhaps some other undesirable property.' (slides p.18.5)

'A procedure is thread safe when the procedure is logically correct when executed simultaneously by several threads' ("Multithreaded Programming Guide". Oracle Corporation. November 2010)

thread1: x := 0, x=x+5

thread2: x := 0, x=x+2

x may not be 5 after thread1 ends.

3. What is liveness? Give a concrete example of a liveness violation.

Liveness = ensuring progress (slides p.19)

e.g. no deadlock or starvation, as threads waiting on each other doing no work.

y := 0, x := 0

thread1: while (x=0) {y = 0} , y = 1 , further progress

thread2: while (y=0) {x = 0} , x = 1 , further progress

4. Using the implementation in the slides, can a binary semaphore lead to a deadlock? Can it lead to starvation?

Explain with the aid of an example.

Starvation: If multiple processes want to access the same data with a binary semaphore, one of the processes may starve if there is not e.g. a queue for fairness.

Deadlock: data1 and data2 both secured by binary semaphors.

process1 accessed and now blocks data1 and wants to access data2 in its critical section

process2 accessed and now blocks data2 and wants to access data1 in its critical section

5. Why do we need synchronization mechanisms in concurrent programs?

slides p.27.5

6. How do monitors differ from semaphores? Please provide a precise answer.

7. How are monitors and message passing similar? And how are they different?

Exercise 2 (2 points)

x := 1

Thread 1 -> x := x + 7.

Thread 2 -> x := x \* 5.

Exercise 3 (1 points)

Implement a monitor using semaphores. Use pseudo-code and comment it