Informatics and Mathematical Modelling
Technical University of Denmark
Building 321
DK–2800 Lyngby
Denmark

HHL 2008–10–03

02152 CONCURRENT SYSTEMS FALL 2008

CP Exercise Class 5

Monday October 6

Monitor Construction

Unless stated otherwise, you should assume (multiple) condition queues with *signal-and-continue* (SC) semantics and without spurious wakeups (ie. the standard semantics used in [Andrews]). Furthermore the predicate $empty(c)$ and the function $length(c)$ on a condition queue c can be used.

1. In Andrews Figure 5.7, a *Timer* monitor is implemented by a *covering condition* which is *true*, ie. whenever the *tod* (time-of-day) is changed, all waiting processes are woken up.

Optimize the monitor such that the processes are only woken up when the next relevant point of time is reached.

2. Are there any differences between semaphores and condition queues?

3. What happens if a `synchronized` Java method contains `while (!b) Thread.sleep(100);`?

4. Let M be a positive constant. Consider the following specification of a *chunk semaphore*:

> **monitor** *ChunkSem*;
>
> **var** s : *integer* := 0;
>
> **procedure** $V()$: $\langle\, s = 0 \rightarrow s := s + M \,\rangle$;
>
> **procedure** $P()$: $\langle\, s > 0 \rightarrow s := s - 1 \,\rangle$;
>
> **end**;

(a) Implement the monitor.

(b) State and argue for a monitor invariant expressing the range of the variable s.

(c) State a monitor invariant expressing that calls of $P()$ do not wait unneccesarily.

(d) Suppose that M is small compared to the number of processes that may call $P()$. Does your solution avoid unneccesary wakeups? If not, try to minimize the wakeups. Is the property from (c) still a monitor invariant? If not, try to remedy this.

5. Do Exercise Mon.3

How should the monitor work if called by more than N processes? Does your monitor work in that case?

6. Determine if your monitor is robust towards *spurious wakeups*.

7. Discuss whether the monitor could be implemented as a Java monitor.