COMP3215
Real-Time Computing and Embedded Systems
Introduction
Denis A Nicole
2014-09-29
Lecture Slots

On average, I will use three slots a week. They will be selected from:

- Mon 09:00 B45/0013
- Tues 10:00 B07/3019
- Wed 09:00 B58/1009
- Thur 16:00 B06/1083

This first week, I will use the Mon, Tues and Wed slots.
Laboratories

- Laboratory sessions will be Wednesdays 10:00–13:00 in three pairs, in the Zepler Electronics laboratory.

- Each lab will consist of an introductory session and an assessment session two weeks later. The lab dates are:

<table>
<thead>
<tr>
<th>Lab</th>
<th>Date 1</th>
<th>Date 2</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 1</td>
<td>15th Oct</td>
<td>29th Oct</td>
<td>Free RTOS/ARM</td>
</tr>
<tr>
<td>Lab 2</td>
<td>5th Nov</td>
<td>19th Nov</td>
<td>Embedded Linux</td>
</tr>
<tr>
<td>Lab 3</td>
<td>26th Nov</td>
<td>10th Dec</td>
<td>DSP</td>
</tr>
</tbody>
</table>

- You will work in randomly assigned pairs for each laboratory.
Textbooks

• Hermann Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications* (second edition) [library online collection]


Syllabus

• Issues and concepts
  – Definition of real-time
  – Temporal and event determinism
  – Architecture review and interfacing
  – Interrupts, traps and events
  – Response times and latency
  – Real-time clocks

• Application domains
  – DSP
  – Safety critical
  – Small embedded
  – Large-scale distributed

• Low-level programming for real-time
  – I/O
  – Concurrency: memory models and synchronisation primitives.
  – Monitors/condition variables
  – Semaphores
  – Optimistic scheduling
  – ARM and Intel assembly language, integration with C.
  – Architectural issues, memory models.

• Scheduling
  – RMS
  – EDF
  – priority inversion
  – Time triggered

• Operating systems
  – Protected modes, virtual memory.
  – Device drivers
  – Internet of things: TinyOS & Contiki
  – FreeRTOS
  – Real-Time Linux

• Languages in real-time
  – Real-time Java
  – C and C++ standards: MISRA

• Correctness
  – Concurrency Issues
  – Process algebras
  – Model checkers, temporal logic
What is a real-time system?

- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period

  - the correctness depends not only on the logical result but also the time it was delivered

  - failure to respond is as bad as the wrong response!

- The computer is a component in a larger engineering system ⇒ EMBEDDED COMPUTER SYSTEM

- 99% of all processors are for the embedded systems market
Compare with conventional systems

- Partial correctness: if a result is returned, it is correct. Established using invariants.

- Total correctness: will terminate and returns a correct result. Troubled by the halting problem.

- Real-time correctness: will return a correct result by the deadline.
Example of partial correctness

• C in a functional style

```c
Integer sumsq(const Integer n) {
    assert(n>=0);
    return (n==0)? 0: sumsq(n-1) + n*n;
}
```

• Obviously returns the correct answer, if it returns at all.

• No variables are changed; so all program values behave like algebraic unknowns

• **NOTE:** I have used a fake type, `Integer`, which does not overflow.
This is also partially correct

- C in a functional style

```c
Integer sumsq(const Integer n) {
    assert(n>=0);
    return (n==0)? 0: sumsq(n+1) – (n+1)*(n+1); }
```

- Obviously returns the correct answer, if it returns at all.

- No variables are changed; so all program values behave like algebraic unknowns

- **NOTE:** I have used a fake type, `Integer`, which does not overflow.
Steps to (non-Real-Time) correctness

1. Use idealised mathematical types to simplify the reasoning.

2. Establish partial correctness.

3. Establish that the function will terminate. The second example does not, if $n>0$.

4. Arrange to cope with any problems from C’s restricted int implementation.
We still have not touched on Real-Time

• The example has time complexity $O(n)$. So would a loop-based implementation; recursion is not a problem here.

• A loop-based function would, however, have constant space complexity. A naïve implementation of the recursive form would use $O(n)$ space.

• A clever implementation could use tail-recursion to reduce space complexity to $O(1)$.

• A good real-time implementation would take time $O(1)$. 