Real-Time and Concurrent Programming

Lecture 1 (F1):

Embedded concurrent software

Klas Nilsson

2016-08-30
Part I

Course Information
Outline

- Following information about the course, today’s lecture will cover:

1. Concurrency
2. Minimalistic concurrent applications
3. Basic concepts
4. Semaphores
5. Threads

- Home-page of the course: http://cs.lth.se/EDA040
- All material/slides in English but remaining lectures in Swedish; Extra session for exchange students, when?
- On the course home-page, you find links to the enrolls for the labs (HT1) and later for the project (HT2 grouping).
Course Information

About this course

Course material

The course program is available via the EDA040 home-page. The combination of printed and electronic course material is under consideration. Also, we recently found some useful additional (electronically available) reading. Therefore, the printed booklet that earlier years was provided at first lecture (at a net/non-profit cost of 100SEK) will not be composed this year.

▶ What is the general interest in printed material? Learning outcome?
▶ Exercises and material for labs are available via the homepage. Bring (e.g., electronically) your course material to the exercise sessions!
▶ Some additional material for labs and for the project will be provided (at no or net cost) during the course.
▶ Practices, updates and reading instructions via web.

The printed booklet version from previous course instances works fine (for those retaking the course/exam).
Ongoing and planned improvements, as of last year:

- Booklet and articles updates.
- Increased emphasis on labs forming milestones.
- No hardware for Lab1.
- C-programming on camera side in the projects.
- Course material better adapted to online reading.
- Labs are still done pairwise, but each individual should maintain/provide their solution.
- Possible addition of state-machines to course content (Lecture 7).

Major revision of written material postponed to next year.
Related courses

The following courses are technically related with this one, with small overlaps in initial content, but with quite different focus.

**EDAN15; Design of Embedded Systems:** Also embedded software with issues of parallelism and scheduling, but towards programmable hardware (VHDL) and System on Chip (SoC) [http://cs.lth.se/EDAN15](http://cs.lth.se/EDAN15) (See also EDA385)

**FRTN01; Real-Time Systems:** Introduces concurrent programming and covers real-time aspects for control purposes, but it is bigger course with the main parts about discrete-time feedback control [http://www.control.lth.se/FRTN01](http://www.control.lth.se/FRTN01)

**EDA050; Operating Systems:** Concurrency and OS processes with resembling techniques, but not targeting embedded systems and the platform rather than application development is the focus. [http://cs.lth.se/EDA050](http://cs.lth.se/EDA050)

**EIEF01; Applied Mechatronics:** Quite short introduction to concurrency and embedded software in line with lab 1, but emphasis is on the combination with electro-mechanical systems and engineering/project aspects. [http://iea.lth.se/tillmek](http://iea.lth.se/tillmek)

There are extended/alternative lab assignments for students studying both this course and FRTN01 or EIEF01. The involved departments collaborate towards coherent software techniques and maximum learning outcome.
## Threads and processes

### Threads in OS-process
- Semaphore object within program/JVM
- Monitor (mutex for shared data)
- Messages (data flow via event buffers)
  - sub-milliseconds
- Hosted in os-process
- Language support
- class Thread, synchronized

### OS-process
- OS-Semaphore as a resource
- File content and file locking
- Pipe, stream and socket
  - milliseconds
- Hosted in computer
- OS/API support
- classes System, Runtime, Process

### Embedded threads
- Sem object and/or HW
- Monitor (on-board memory)
- Messages (network, fieldbus, event buff)
  - microseconds
- Hosted on board
- Language support
- class Thread, synchronized

---

EDA040 practice

See EDA050

EDA040 content

[http://cs.lth.se/EDA040](http://cs.lth.se/EDA040)
Embedded computing for you as a consumer

Software everywhere

- Most 'chips' programmed; you use a lot of software.
- Embedded microprocessors by far the most common type of computer.
- Scarce computing resources for market competitiveness.
- Should not behave (or crash) like desktop PCs.
Embedded computing for you as an engineer

Likely roles for you in industry:

- Developer of core platforms
- Application engineering
- Technical support
- Sales or Purchasing (components, subsystems, outsourcing, ..)
- Management (of products, projects or departments)
- New company (systems and/or solutions)

The software technologies and the associated predictability/reliability issues we are covering in this course is highly relevant for all these professional roles.
Products with embedded software
Embedded control imposes real-time requirements

Control

Computer software

Communication

Actuators

Sensors

Mechanics

Domestic robot

Cowbot

Medical

Combustion engine

Robot

Component
Distributed software and real-time networking

Volvo XC 90:
- 3 CAN-buses
- + other buses

40-100 ECUs in a new car
~ 2-5 million lines of code
Part II

Embedded concurrent real-time software
1 Concurrency

2 Minimalistic concurrent applications

3 Basic concepts

4 Semaphores

5 Threads
Correctness

The software must

- perform computations logically correct in each activity, and
- react on input events concurrently and give the correct output for any possible execution order, otherwise the systems may fail (is not correct).

Each activity should be a *reactive*; responding to time increments and external events, but not executing when nothing happens.
Each concurrent activity needs to be a logically correct sequential program.

All concurrent activities together must be concurrently correct.
Concurrency and software constructs

Application concurrency
- Sequential computing within activity

Programming means
- Normal Java (or C/C#/..) code/execution
Concurrency and software constructs

Application concurrency

- Sequential computing within activity
- Parallel/physical environment

Programming means

- Normal Java (or C/C#/..) code/execution
- Concurrent threads (or processes or tasks)
Concurrent and software constructs

**Application concurrency**

- Sequential computing within activity
- Parallel/physical environment
- Data and operations

**Programming means**

- Normal Java (or C/C#/..) code/execution
- Concurrent threads (or processes or tasks)
- Object-Oriented Progr.

http://cs.lth.se/EDA040
Concurrency and software constructs

Application concurrency

- Sequential computing within activity
- Parallel/physical environment
- Data and operations
- Mutual exclusion; critical sections

Programming means

- Normal Java (or C/C#/..) code/execution
- Concurrent threads (or processes or tasks)
- Object-Oriented Progr.
- Mutex-semaphore or synchronized
Concurrent and software constructs

Application concurrency

- Sequential computing within activity
- Parallel/physical environment
- Data and operations
- Mutual exclusion; critical sections
- Signaling between activities

Programming means

- Normal Java (or C/C#/..) code/execution
- Concurrent threads (or processes or tasks)
- Object-Oriented Progr.
- Mutex-semaphore or synchronized
- Counting-semaphore or wait/notify

http://cs.lth.se/EDA040
F1: Embedded concurrent real-time software
Concurrency and software constructs

**Application concurrency**

- Sequential computing within activity
- Parallel/physical environment
- Data and operations
- Mutual exclusion; critical sections
- Signaling between activities
- Response time requirements

**Programming means**

- Normal Java (or C/C#/..) code/execution
- Concurrent threads (or processes or tasks)
- Object-Oriented Progr.
- Mutex-semaphore or synchronized
- Counting-semaphore or wait/notify
- Real-time scheduling, scheduling analysis
Our focus now: The upcoming first lab

- The first half of the lectures the focus is on introducing the labs.
- The first five exercises also do that, either as direct preparation or by introducing the concepts.

<table>
<thead>
<tr>
<th>Exercise 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Semaphores</td>
</tr>
<tr>
<td>2. Signaling and mutex</td>
</tr>
<tr>
<td>3. Place operations inside methods and objects.</td>
</tr>
</tbody>
</table>

Basics for Lab 1.

<table>
<thead>
<tr>
<th>Lab 1 (&amp; exercise 2):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use semaphores to implement software for an alarm clock.</td>
</tr>
<tr>
<td>Event-driven and time-driven threads sharing time data.</td>
</tr>
</tbody>
</table>

Essence of threads and semaphores.
1 Concurrency

2 Minimalistic concurrent applications

3 Basic concepts

4 Semaphores

5 Threads
The Lego-brick machine

Implementation without concurrent programming support

while (true) {
    while (! isEmpty) {
        tempControl();
        sleep(dT);
    }
    On(doPush);
    while (! isOut) {
        tempControl();
        sleep(dT);
    }
    off(doPush); on(doPull);
    while (! isIn) {
        tempControl();
        sleep(dT);
    }
    off(doPull);
}

Using single activity of the main program

- Control temperature (periodically) and the piston (event driven).
- As one program/activity (note the code duplication):
Improved structure by concurrent programming

// Activity 1: Temperature
while (true) {
    if (temp > max)
        off(doHeat);
    else if (temp < min)
        on(doHeat);
    sleep(dT);
}

// Activity 2: Piston
while (true){
    await(isEmpty);
    on(doPush);
    await(isOut);
    off(doPush);
    on(doPull);
    await(isIn);
    off(doPull);
}
Correct concurrency

Consider bank account transactions by you (A) and your employer (B)

**Case 1: Withdraw first**

A: Read 5000  
A: amount = 5000 - 1000  
A: Write 4000  
B: Read 4000  
B: amount = 4000 + 10000  
B: Write 14000

**Case 2: Salary first**

B: Read 5000  
B: amount = 5000 + 10000  
B: Write 15000  
A: Read 15000  
A: amount = 15000 - 1000  
A: Write 14000

Two activities (threads or processes) independently make their transactions, but not at the same time. Correct result in both cases.
Faulty concurrency

With run-time support for concurrent activities, we might get

Timing 1:

A: Read 5000
B: Read 5000
A: amount = 5000 - 1000
   B: amount = 5000 + 10000
A: Write 4000
   B: Write 15000

Timing 2:

A: Read 5000
B: Read 5000
   B: amount = 5000 + 10000
B: Write 15000
A: amount = 5000 - 1000
   A: Write 4000

- Concurrency fault: Wrong result for some interleavings.
- Needed: Mutual exclusion (critical sections, atomic actions)
Observations and reflections

The Lego-brick machine

- For the temperature control the loop is timed (timing?).
- The piston control is a sequencing control.
- Concurrency control is about ensuring machine performance.

The bank account

- Sequencing is due to computer/machine operations as such.
- Concurrency control is about maintaining system consistency.

The concurrent activities for the ..

- Lego-brick machine are only connected physically.
- Bank account are only connected via shared data.
1. Concurrency

2. Minimalistic concurrent applications

3. Basic concepts

4. Semaphores

5. Threads
You know that the computer works sequentially, but:

- Consider the following code snippet:

```c
int x = 2;
while (x>1) {
    if (x==10) x = 0;
    x++;
};
```
You know that the computer works sequentially, but:

- Consider the following code snippet:

```c
int x = 2;
while (x>1) {
    if (x==10) x = 0;
    x++;
}
```

- Easy: What value will x get?
You know that the computer works sequentially, but:

- Consider the following code snippet:

  ```java
  int x = 2;
  while (x>1) {
    if (x==10) x = 0;
    x++;
  }
  ```

- Easy: What value will `x` get?
- Tricky: Why do many beginners give the answer 0?
You know that the computer works sequentially, but:

- Consider the following code snippet:

```java
int x = 2;
while (x>1) {
    if (x==10) x = 0;
    x++;
}
```

- Easy: What value will x get?
- Tricky: Why do many beginners give the answer 0?
- Answer: They do not think sequentially yet.
Sequential and parallel behaviors/objects

Sequential activities:
- Single actor/user actions.
- Software execution.

Inherently parallel:
- Physical dynamics.
- Electronic hardware.

but sequenced by IO system.

Concurrent programming: Use multiple sequences/activities to manage multiple actors and IO from/to a parallel environment.

Embedded computers; control software: Sequential programs behaving concurrently and in real time to control a parallel environment. Control actions impose real-time requirements.

⇒ We must think both sequentially and concurrently...
Activity – Process – Thread

Definitions concerning concurrently running (swe: jämlöpande) software:

**Activity**  An entity$^0$ performing$^1$ actions$^2$.

**Process**  An entity, holding its own resources, performing$^1$ instructions$^3$, typically in a competing manner.

**Job**  Sequential instructions to reach a well-defined state, such as the initial state, to be performed by an activity, within a given time with given resources and without interference of other activities.

**Task**  A set of jobs being sequentially performed by some process.

**Thread**  A sequential activity that performs instructions.

---

$^0$That which is perceived or known or inferred to have its own distinct existence

$^1$Actively doing, in a sequential fashion, possibly concurrently.

$^2$Something being done/made/performed.

$^3$A description of how to perform a set of actions.
1. Concurrency

2. Minimalistic concurrent applications

3. Basic concepts

4. Semaphores

5. Threads
Critical sections

- Parts of a program that access a shared resource/object
- May not be interrupted by each other, or by another invocation of itself, *for that shared data/object*.
- Accomplished by Semaphores, Monitors, or Mailboxes.
- In low-level/native code, such as device drivers, hardware interrupts can be disabled.

Well known concern in software, see for instance http://en.wikipedia.org/wiki/Critical_section

In general terms, a resource need to be *locked* for exclusive access.
## Locking the bank account for mutual exclusion

Temporarily locking the account (but not the bank) over *critical section*.

### Case 1: Withdraw first

A: Lock account // First; no blocking
A: Read 5000
    B: Lock account // Blocking
A: amount = 5000 - 1000
A: Write 4000
A: Unlock // Will unblock B
    B: Complete locking
    B: Read 4000
    B: amount = 4000 + 10000
    B: Write 14000
    B: Unlock

### Case 2: Salary first

B: Lock account // First; no block
B: Read 5000
A: Lock account // Blocking
    B: amount = 5000 + 10000
    B: Write 15000
    B: Unlock
A: Complete locking // End blocking
A: Read 15000
A: amount = 15000 - 1000
A: Write 14000
A: Unlock

- Except the four lines with italic font, threads A&B are ready or running, managed by OS scheduler.
- The “Complete locking” requires that thread to be scheduled; with strict priorities a thread C could pass and lock.
Semaphore – A minimal mechanism for synchronization

Positive integer variable with two operations, take and give:

```java
class SemaphorePrinciple {
    int count;
    public void take() {
        while (count < 1) "suspend execution of caller";
        --count; // Got the semaphore, continuing ....
    }
    public void give() {
        if ("anyone suspended") "resume the first in queue";
        count++;
    }
}
```

NOTE: take and give are 'atomic' (odelbara) operations which require system support to implement, e.g. by disabling hardware interrupts or using a Test-and-Set instruction.

NOTE: In take, as long as count==0 the caller is blocked, i.e. the execution is stopped, which differs from methods that can be implemented by ordinary Java code.
Java — Mutex Semaphore

- Declaration

```java
import se.lth.cs.realtime.semaphore.*;
Semaphore mutex1; // Not that good/clear.
MutexSem mutex2; // Better, more expressive.
```

- Create, initialize.

```java
/* Possible but not recommended: */
mutex1 = new CountingSem(); // Assigns the value zero.
mutex1 = new CountingSem(1); // Assigns the value 1.
/* Preferred: */
mutex2 = new MutexSem(); // Assigns 1 to internal state.
```

- Use

```java
mutex2.take();
amount += change; // Bank account transaction in critical section
mutex2.give();
```

- Apart from clarity, using a MutexSem for mutual exclusion permits better efficiency and error detection.
Semaphore – usage

**Mutual exclusion**

// Thread A
mutex.take();
***
mutex.give();

// Thread B
  ●
  mutex.take();
  ***
  mutex.give()

**Signaling**

// Thread A:
  ●
  elements.give();
  ●

// Thread B:
  ●
  elements.take();
  ●
<table>
<thead>
<tr>
<th></th>
<th>Concurrency</th>
<th>Minimalistic concurrent applications</th>
<th>Basic concepts</th>
<th>Semaphores</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Concurrency</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Minimalistic</td>
<td>concurrent applications</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Basic concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Semaphores</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Threads</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution states and semaphores

A Thread is concurrently executing within a program, bore on that next lecture

Scheduling states

- Running
- Ready
- Blocked

Required for first lab:

- You have to use Semaphores appropriately in your code such that is gets concurrently correct!
- Proper blocking (only Ready/Running when needed).