E7020E Embedded System Design 2021

SW-1 The Art of Embedded Programming

Per Lindgren

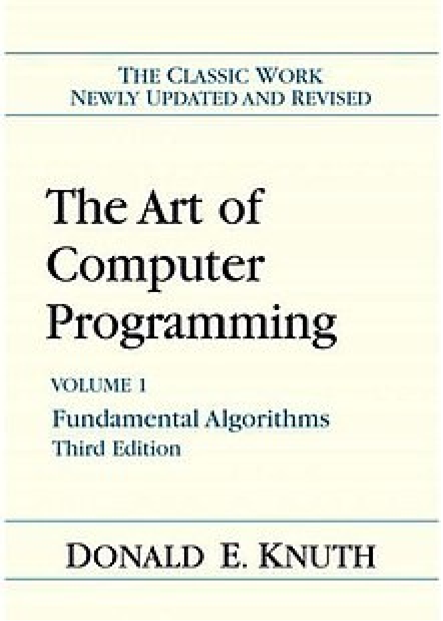
Professor Embedded Systems, DCC/LTU

Computer Science

- Theoretical Computer Science
  - applies the principles of **mathematics** and **logics**, to develop
  - **correct** and **efficient** solutions to problems
  - **efficient** in term of **complixity** measured by number of operations (computatinons, memory accesses), total memory required, etc.

Theoretical Computer Science
The Art of Computer Programming

- Donald Knuth
  - Vol 1. (1962 -) Fundamental algorithms
  - Vol 2. Seminumerical algorithms
  - Vol 3. Sorting and Searching
  - Vol 4. Combinatorial algorithms
  - Vol 5-7 (- 2021) Ongoing



THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of
Computer
Programming

VOLUME 1
Fundamental Algorithms
Third Edition

DONALD E. KNUTH

- Algorithms and data structructures
  - MIX assembly language (for “cost” of operations)

- Turing award 1974 (analysis of algorithms)

Computer Science
Software Engineering

- Software Engineering
  - applies **languages** and **type** systems to develop
  - **correct** and **efficient** solutions to problems
  - **efficiency** take aspects in mind such as
    - **complexity**, but also
    - **lifecycle** management
      (development, deployment, maintenance, etc.)
    - **re-use**
    - **scalability**

Computer Science
Computer Engineering

- Computer Engineering
  - applies **computer science** and **electronic engineering** to develop
  - **correct** and **efficient** solutions to problems
  - **correctness** and **efficiency** are multi-disciplinary
    - Functional properties (computed values and output)
    - Non-functional
      - **Safety** properties (what a system may never ever do)
      - **Liveness** properties (what a system must eventually do)
      - **Timely properties** (e.g, output vs time, ordering etc.)
      - **Power consumption**
      - **Physical size**
      - **Production cost**
      - **Security**

Computer Science/Engineering
The Art of Embedded Programming

- Embedded Programming
  - The functionality of the system relies to an increasing extent on embedded software
  - It is a fundamentally hard problem, recall:
  - **correctness** and **efficiency** are multi-disciplinary
    - Functional properties (computed values and output)
    - Non-functional propreties, the firmware has implications to
      - **Safety** properties (what a system may never ever do)
      - **Liveness** properties (what a system must eventually do)
      - **Timely properties** (e.g, output vs time, ordering etc.)
      - **Power consumption**
      - **Physical size**
      - **Production cost**
      - **Security**

Computer Science/Engineering The Art of Embedded Programming

- Embedded Programming
  - However
    - The software engineer don’t get the hardware design
    - The hardware engineer don’t get the software design
  - And more alarming
    - Lack of methodology, leading to
    - Ad-hoc solutions
    - Trial and error
    - Yeeey, now it works, don’t touch it!!!

Computer Science/Engineering
The Art of Embedded Programming

- Embedded Programming
  - State of Practice
    - C programming (C++ often used in C mode)
    - Vendor specific libraries and tools, e.g.
      - STM32 Hardware Abstration Layer (HAL)
      - CubeMX
  - Inherently **unsafe** access to memory
    - Hard/impossible to prove correctness
    - Hard/impossible to ensure security

Computer Science/Engineering
The Art of Embedded Programming

- Embedded Programming
  - Why C/C++?
    - Allows to take fine grained control over HW
    - Memory and CPU efficient binaries
    - Predictable execution (what you C is what you get)
  - Lack of **mainstream** alternatives providing
    - Fine grained control over HW
    - Memory and CPU efficient binaries
    - Predictable execution

Computer Science/Engineering The Art of Embedded Programming

- Domain Specific Languages (DSL)
  - PLC 1131/61499 (industrial control systems)
  - Labview (compiles to C, industrial monitoring/control)
  - Matlab/Simulink (compiles to C, control systems)
  - Erlang (telecom systems)
  - Signal/Esterel/Lustre (synchronous programming, safety critical)
  - Ada Sparc (for proofs over programs, safety critical)
- **Not mainstream**
  - Limited support for MCUs
  - Requires expert knowledge

Computer Science/Engineering
The Art of Embedded Programming

- Embedded Programming
  - Rust, the “least bad” language
    - Fine grained control over HW
    - Memory and CPU efficient binaries
    - Predictable execution
  - Built in memory safety, beneficial to
    - **correctness**
    - **security**

Computer Science/Engineering
The Art of Embedded Programming

- Rust
  - Raw memory access requires explit ***unsafe*** code
    - Allows you to do all the “dirty stuff” you need,
      but correctness (soundness) is on you
  - “**Zero-cost**” abstractions used to
    - Hide the dirty stuff from the end user
    - Provide a “fail safe” API
  - Allows us to “single out” dangerous code
    Done right the compiler will prevent misusage

Zero-cost in Rust does not imply no-overhead at all,
it merely implies that the implementation overhead is
dictaded by the problem at hand,
i.e., doing it manually would have the same/similar overhead

Computer Science/Engineering The Art of Embedded Programming

- Embedded programming is still a mess
  - How to deal with concurrency
    - Race conditions
    - Deadlocks
    - Timing predictability

Computer Science/Engineering The Art of Embedded Programming

- Concurrency and parallelism
  - Batch jobs (1950-60, recall the “punch card” piles?)
  - Multipile piles (parallelism)
    - Processes on a Multi-Processor computer (1960-70)
    - Context switch between processes costly
  - Threads, a light weight context inside a process
    - Context switch between threads less costly
    - IBM OS/360

An implementation technique to reduce cost

Computer Science/Engineering
The Art of Embedded Programming

- Threads come in different flavours, e.g.,
  - POSIX Threads (pthreads)
    - > 50 primitives
    - Each with different options
  - In general:
    - You cannot determine the behavior by looking localy on the code
    - E.g., mutex behavior depends on:
      - The chosen scheduling policy
      - How the mutex was created
      - Other threads setting attributes at run-time
      - Not only the code that uses the mutex

Computer Science/Engineering The Art of Embedded Programming

- In effect, threads are
  - NOT suitable to model concurrency
  - merely a complex and costly way to implement concurrency control
  - extremely hard to get right, race-conditions, deadlocks, poisioning, etc. etc.

Computer Science/Engineering
The Art of Embedded Programming

- Real-time systems in literature are typically modelled in terms of:
  - **tasks** with shared **resources**
  - **communicating processes** (message passing/actors)
- These models are well understood/researched

Computer Science/Engineering The Art of Embedded Programming

- An example:
  Stack-Based **R**esource Allocation **P**olicy for Realtime Processes
  Baker 1990:

  - **SRP** provides an efficient means to sheduling single-core/processor systems:
    - **Tasks** with shared **Resources**
    - Task are **run-to-completion**
    - Resources must be taken in **LIFO** order

  - **SRP** brings:
    - Race- and Deadlock-free execution
    - Single blocking (bounded priority inversion)
    - Single stack execution
    - Theory for schedulability and response time analysis

Computer Science/Engineering The Art of Embedded Programming

- An example:
  Schedulability of asynchronous real-time concurrent objects
  Jagori et. al. 2009:
  - Actor model:
    - Local state
    - Message passing
  - Actor model brings:
    - Race- and Deadlock-free execution
    - Theory for schedulability and response time analysis (stated and solved as timed-automata problem)

Computer Science/Engineering The Art of Embedded Programming

- Why still threads?
  - Embedded Linux
  - QNX
  - VxWorks
  - FreeRTOS
  - ChibiOS
  - Etc.
- Well... why still C?

  The answer is the same

  - Lack of viable alternatives!

Computer Science/Engineering
The Art of Embedded Programming

RTIC

Real-**T**ime **I**nterrupt-driven **C**oncurrency

Computer Science/Engineering
The Art of Embedded Programming

- RTIC
  Combines synchroneous and asynchroneous modelling
  - Adheres to the Stack Resource Policy (intra-core)

    This allows for synchroneous resource access

  - Adheres to the Actor message passing model (intra- and inter-core)

    This allows for asynchroneous programming

  - Implemented in Rust
    (the framework)

  - Implemented for Rust
    (the user code in Rust)

Computer Science/Engineering The Art of Embedded Programming

- Embedded Programming is still a mess
  - Rust is young, v1.0 released 2015
    - Language under constant development
  - Embedded Rust Ecosystem is still young
    - Most parts not yet v1.0
    - Abstractions and tools under development
  - RTIC is young
    - Not yet v1.0
    - Task/resource/actor model not yet commonplace
  - Join the party, we need You
    - Skilled software engineers that understand hardware
    - Skilled hardware engineers that understand software
  - Embedded Rust + RTIC already in used in production
    - Products on the market since 2018

Computer Science/Engineering
The Art of Embedded Programming

- Knuth still works on
  “The Art of Computer Programming”

  > 50 years in the making...

- Let’s go for

  “The Art of Embedded Programming”

  Vol 1: C free programming
  Vol 2: Where are my threads?
  Vol 3: Let’s build a mouse together

  (By the way it should be finished by March 27th)