IS1330 Real-Time Systems
Lecture 1

Teachers:
Olle Bowallius
Phone: 790 44 42
Email: ollebo@kth.se
Bengt Molin
Phone: 790 44 48
Email: bengtm@kth.se

Aim of the Course
That the student should be able to design a real-time system. Thus for a passed
grade the student should be able to:
- summarize a design method for a real-time system
- explain how to handle exceptions and how to design reliable real-time systems
- apply concurrent programming by using operating systems, programming
  languages and software libraries
- apply methods for handling critical sections in a system with parallel processes
- explain the primary characteristics and the main functionality of a RTOS
- explain different methods for scheduling of processes and make utilization-
  based schedulability tests
- identify suitable transducers for a specific case and use them in a real-time
  system

Course Material
- Burns, A., and A. Wellings, Real-Time Systems and Programming Languages,
- Manual for microcontroller and Lentz Digital Interface
- Roco Model Railroad (2st).
- Webpage

Supplementary rules for examination of all courses at the IT University
- Regulation 1: All group members are responsible for group assignments
- Regulation 2: Recount correctly any help received and sources used
- Regulation 3: Do not copy the solutions of others
- Regulation 4: Be prepared to present your solution
- Regulation 5: Use the attendance list correctly

Course Overview

Concepts
- Batch-, Interactive- and Real-Time Systems
- Hard and Soft Real-Time Systems
- Criticality
- Concurrent programming
- Reliability, Risk and Fault-Tolerance
Categorizing Computer Systems

- Batch Systems: I don’t mind when the computer results arrive, within reason.
- Interactive on-line: I would like the results within a fairly short time, typically a few seconds
- Real-time: I need the results within definite timescales, otherwise the systems just won’t work properly.

[Cooling, 2003]

Examples of Real-Time Systems

- Vehicle systems
- Traffic control
- Process control
- Medical systems
- Military uses
- Manufacturing systems with robots
- Telecommunications
- Computer games
- Multimedia systems
- Household systems
- Building managers
- Often component of a large engineering system, i.e. an embedded system
- 99% of all processors are for embedded systems!

[Burns and Wellings, 2001]

Definition of Real-Time Systems

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 at the time it was delivered.

[Burns and Wellings, 2001]

Characteristics of RTS

- Timing constraints
- Criticality
- Concurrency
- Reliability and fault-tolerance
- Application specific

[Burns and Wellings, 2001]

Timing Constraints

Reactive system
- System behaviour is caused by arrival of aperiodic events

Time-driven system
- System behaviour is driven by periodic tasking
- Hard deadlines are performance requirements that must be met
- Soft real-time systems are those where the system works correctly even if deadlines occasionally are missed

Timing Characteristics

Hard deadlines
- Late results are bad results.

Soft deadlines
- Late data may still be good data

Firm deadlines
- Late data is worthless results
Timing constraints

Real-Time Systems

- The important part is to know when things happen, not necessary do very fast calculations.
- Real-time systems must fulfill timing constraints.
- High performance computer systems are optimized regarding average throughput.

Airbag system

- A car is about to crash into a tree.
- The car has an airbag.

Car crashes into the tree
Guy hits head into the wheel

Criticality

Critical

- Stock exchange
- Aircraft Flight control

Soft
- Multimedia
- Computer game

Hard

Non critical

[Shaw, 2001]

Concurrency

- Most real-time applications are inherently parallel:
  - External elements interacts with real-time systems in parallel. Concurrent programming can be used to model this parallelism.
  - Structuring a concurrent program into parallel processes results in separation of concerns.
  - Structuring a system into concurrent processes can result in an overall reduction in execution time.
  - Concurrent processes can be given different scheduling priorities dependent on how time-critical its functions are.

- Mapping these concurrent activities to one sequential program leads to a more complex design.
  - The programmer must design the system so that it involves the cyclic execution of a program sequence.
  - This complicates the programmer’s already difficult task.
  - It makes decomposition of the problem more complex.
  - The placement of code to deal with faults is more problematic.
  - Parallel execution of the program on more than one processor will be even more difficult to achieve.

Reliability and Fault-tolerance

- Many real-time systems have high demands for fault tolerance and reliability:
  - Aircrafts
  - Nuclear power plants
  - Industrial automation
  - Medical systems
  - Other devices (games)

[Burns and Wellings, 2001]
Reliability

- **Reliability** is a measure of how well a system operates correctly.
- Can be expressed as mean-time between failures (MTBF).
- **Failure rate** = 1/MTBF.
- **Mean time to repair** (MTTR) gives availability.
- Availability = MTBF/(MTBF+MTTR).

Risk

- Risk depends on:
  - The severity of any failure
  - The probability of failure occurring

<table>
<thead>
<tr>
<th>Probability of failure</th>
<th>Reassuring</th>
<th>Probable</th>
<th>Likely</th>
<th>Extreme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptable</td>
<td>10^-5</td>
<td>10^-4</td>
<td>10^-3</td>
<td>10^-2</td>
</tr>
<tr>
<td>Rare</td>
<td>10^-3</td>
<td>10^-2</td>
<td>10^-1</td>
<td>10^0</td>
</tr>
<tr>
<td>Inprobable</td>
<td>10^-1</td>
<td>10^0</td>
<td>10^1</td>
<td>10^2</td>
</tr>
<tr>
<td>Catastrophic</td>
<td>10^0</td>
<td>10^1</td>
<td>10^2</td>
<td>10^3</td>
</tr>
</tbody>
</table>

Application Specific

- The software for real-time systems are often tailor-made or adapted for the particular application.
- The system must interact with the external world via specific sensors and actuators.

Concepts

- Batch-, Interactive- and Real-Time Systems
- Hard and Soft Real-Time Systems
- Criticality
- Concurrent programming
- Reliability, Risk and Fault-Tolerance

Assignment

- Find 5 things in your daily life which contains a computer and determine if you think they are:
  - Batch, Interactive or Real-Time Systems
  - Hard, Soft or Firm Real-Time Systems
  - High or Low Criticality Systems
  - High or Low Reliability Systems
  - Fault-Tolerant Systems
  - Concurrent Systems
Concepts

- Software Engineering
- Life-cycle model
- Requirement, Analysis, Design, Implementation, and Postdelivery maintenance
- Iterations, Incremental
- Cohesion and Coupling
- Object-oriented development

Scope of Software Engineering

- Historical Aspects
  - Efforts to solve the “Software Crisis”
  - Software is delivered
    - Late
    - Over budget
    - With residual faults

Outcome of 9000 projects completed during 2004

- Project Outcomes
  - Completed
  - Successful
  - Completed late, over budget
  - With residual faults

Life-cycle model

1. Requirements phase
2. Analysis (specification) phase
3. Design phase
4. Implementation phase
5. Postdelivery maintenance phase
6. Retirement

Waterfall lifecycle

- Advantages
  - Documentation
  - Simplify post delivery maintenance
- Disadvantages, compare to
  - Building a house
  - Tailored suit

Real Lifecycle

- Requirements
- Analysis
- Design
- Implementation
Approximate Relative Cost of Each Phase

- 1976–1981 data
- Maintenance constitutes 67% of total cost

Good and Bad Software

- Good software is maintained—bad software is discarded
- Different types of maintenance
  - Corrective maintenance [about 20%]
  - Enhancement
    - Perfective maintenance [about 60%]
    - Adaptive maintenance [about 20%]

Cost to Detect and Correct a Fault

Team Programming Aspects

- Hardware is cheap
  - We can build products that are too large to be written by one person in the available time
- Teams
  - Interface problems
  - Meetings

Communication problems

- Number of communication channels with three team members
- Number of communications channels with one additional team member

The cost of software as a percentage of total cost (hardware+software)
Projects

Design of Computer

A highly incompetent computer architect decides to build an ALU, shifter and 16 registers with AND, OR, and NOT gates, rather than NAND or NOR gates.

Design of Computer (contd)

Design with one gate type per chip
Resulting "masterpiece"

Design of Computer (contd)

Architect designs 3 silicon chips

Cohesion

1. Coincidental cohesion
2. Logical cohesion
3. Temporal cohesion
4. Procedural cohesion
5. Communicational cohesion
6. Functional cohesion
7. Informational cohesion

The Object-Oriented Paradigm

The structured paradigm had great successes initially
- It started to fail with larger products (> 50,000 LOC)
- Maintenance problems (today, up to 80% of effort)
- Reason: structured methods are
  - Action oriented (finite state machines, data flow diagrams); or
  - Data oriented (entity-relationship diagrams, Jackson's method);
- But not both
The Object-Oriented Paradigm (contd)

- Both data and actions are of equal importance
- Object:
  - Software component that incorporates both data and the actions that are performed on that data
- Example:
  - Bank account
    - Data: account balance
    - Actions: deposit, withdraw, determine balance

Structured versus Object-Oriented Paradigm

- Information hiding
- Responsibility-driven design
- Impact on maintenance, development

Key Aspects of Object-Oriented Solution

- Conceptual independence
  - Encapsulation
- Physical independence
  - Information hiding
- Impact on development
  - Physical counterpart
- Impact on maintenance
  - Independence effects

Transition From Analysis to Design

<table>
<thead>
<tr>
<th>Structured Paradigm</th>
<th>Object-Oriented Paradigm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Requirements phase</td>
<td>1. Requirements phase</td>
</tr>
<tr>
<td>2. Specification (analysis) phase</td>
<td>2'. Object-oriented analysis phase</td>
</tr>
<tr>
<td>3. Design phase</td>
<td>3'. Object-oriented design phase</td>
</tr>
<tr>
<td>4. Implementation phase</td>
<td>4'. Object-oriented programming phase</td>
</tr>
<tr>
<td>5. Integration phase</td>
<td>5. Integration phase</td>
</tr>
<tr>
<td>7. Retirement</td>
<td>7. Retirement</td>
</tr>
</tbody>
</table>

- Structured paradigm:
  - Jolt between analysis (what) and design (how)
- Object-oriented paradigm:
  - Objects enter from the very beginning

Concepts

- Software Engineering
- Life-cycle model
- Requirement, Analysis, Design, Implementation, and Postdelivery maintenance
- Iterations, Incremental
- Cohesion and Coupling
- Object-oriented development

Use-case diagram
Estimate Payments and Grants for Week

Use Case

- MSG Foundation

- MSG Staff Member

Estimate Funds Available for Week

Estimate Payments and Grants for Week

Types of Classes

- Entity classes
  - Models long-lived information
- Boundary classes
  - Models the interaction between the product and the environment
  - A boundary class is generally associated with input or output
- Control classes
  - Models complex computations and algorithms

Scenario

- Buy a Product

  - Main Success Scenario:
  1. Customer browses catalogue and selects items to buy.
  2. Customer goes to checkout.
  3. Customer fills in shipping and order information (next-day or 3-day delivery).
  4. System presents full pricing information, including shipping.
  5. Customer fills in credit card information.
  7. System confirms sale immediately.
  8. System sends confirming e-mail to customer.

  - Extensions:
    3a: Customer is regular customer:
      1. System displays current shipping, pricing, and billing information.
      2. Customer may accept or override these defaults, whereas to MSS at step 6.
    6a: System fails to authorize purchase:
      1. Customer may re-enter credit card information or may cancel.

Object-oriented design

- Complete the class diagram
- Perform the detailed design
Implementation

- Good programming practice
- Use Consistent and Meaningful Variable Names
- Comments to help maintenance programmers
- Constants
- Indentation and structure
- Coding conventions

Prologue Comments

- The name of the code artifact
- A brief description of what the code artifact does
- The programmer's name
- The date the code artifact was coded
- The date the code artifact was approved
- The name of the person who approved the code artifact
- The arguments of the code artifact
- A list of the name of each variable of the code artifact, preferably in alphabetical order, and a brief description of its use
- The names of any files accessed by this code artifact
- The names of any files changed by this code artifact
- Error-handling capabilities
- The name of the file containing test data (to be used later for regression testing)
- A list of each modification made to the code artifact, the date the modification was made, and who approved the modification
- Any known faults