Real-Time Systems

Insup Lee
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
www.cis.upenn.edu/~lee/

CIS 480, Spring 2009

Acknowledgement

- Slides are borrowed and/or adapted from the following people:
  - Doron Peled
  - P.S. Thiagaranjan
  - Jane Liu
  - Sebastian Fischmeister

What are Real-Time Computer Systems?

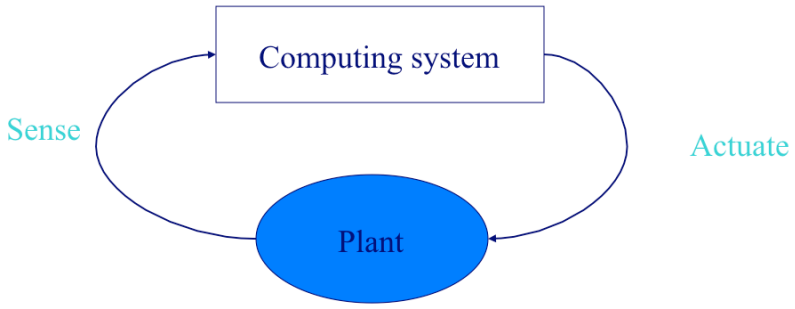
- Real-Time Computer System
  - Correct functioning depends on:
    - the values of the results produced.

    AND
    - the physical times at which the results are produced.
- Real-Time ***Computer*** System is embedded in a larger physical system.

What are real-time systems?

- The state of the system evolves with time.
  - Position, velocity, acceleration
  - Pressure, temperature, level, concentration
- The state needs to be sensed and controlled by the computer system.

Control Function



Real time

- System time and external physical time are the same!
- At least must have a predictable relationship
  - bounded skew, bounded clock rates, etc.

Timing Constraints on computation

- The computing system must sense, compute and actuate in a timely fashion.
- Many sensors and many actuators.
- Many control functions!
- Must schedule computational tasks for different control functions in a timely fashion.
- Computations must finish on time.
  - QoS
  - End-to-end timing constraints

Hard and Soft Real-Time Tasks

Task: a unit of work - a granule of computation, a unit of transmission

Hard real-time tasks –

- Critical tasks, deterministic tasks: failure of a task to complete in time ⇒ a fatal error
- Timing constraints must meet always
- Reactor control, automotive electronics

Soft real-time tasks:

- Essential tasks, statistical tasks: such a task is completed always even when it requires more than available time.
  - Examples: display updates, interpretation of operator commands
- Non-essential tasks: such a task may be aborted if it cannot be completed in time.
  - Examples: connection establishment, monitoring non-critical changes
- Transaction Processing system, Multi-media streaming applications.

Soft vs. hard real-time systems

- Hard real-time systems are typical embedded systems.
- Determined externally

Desired Characteristics of Hard Real Time Computing Systems

- Timeliness
- Peak Load Handling
  - The system should not fail at peak load conditions
- Predictability (not speed, fairness, etc.)
- Fault Tolerance
- Maintainability

Impact on System Architecture

- Must avoid non-determinism (why?)
- Sources of non-determinism:
  - Direct Memory Access (DMA) by peripheral devices
    - Contention for system bus
  - Cache
  - Interrupts generated by I/O devices
  - Memory Management (paging)
  - Dynamic data structures, recursion, unbounded loops (language level)

Examples of real-time applications

- On-line transaction systems and interaction systems
- Real-time monitoring and signal processing systems
  - Typical computations
  - Timing requirements
  - Typical architectures
  - E.g., Railway Switching Systems
- Control systems
  - Computational and timing requirements of direct computer control
  - Hierarchical structure
  - Intelligent control
  - E.g., Chemical and Nuclear Plant Control, flight control
- Embedded systems
  - Resource limitation
  - E.g., Automotive applications

Current State

- Ad hoc techniques, heuristic approaches.
- Code written in C, assembly language
- Programmed timers
- Low level device handling
- Direct manipulation of task and interrupt priorities.
- Goal: Optimized predictable execution on simple architectures.

Drawbacks

- Tedious programming
  - Code quality depends on the programmer
- Difficult to understand, maintain, and reuse
- Verification/testing of timing constraints is practically impossible
- System could collapse in rare and unforeseen circumstances leading to disasters

Laws of Real Time Systems [Buttazzo]

- If something can go wrong, it will go wrong. (Murphy's law)
- Any software bug will tend to maximize damage
- The worst software bug will be discovered 6 months after the field test
- A system will stop working at the worst possible time
- Sooner or later the worst possible combinations of circumstances will occur

Concepts, methods, and Techniques

- Formal methods
- Time triggered architecture
- Real-time scheduling
- Feedback in computer systems
- Assurance cases
- Etc.

Formal Methods (The Ideal!)

- Model real time systems precisely
  - External events
  - System events
- Verify timing properties
- Propagate timing constraints down to the system level
- Verify implementation meets the specification at each level

What are formal methods?

Techniques for analyzing systems, based on some mathematics.

This does not mean that the user must be a mathematician.

Some of the work is done in an informal way, due to complexity.

Examples for FM

Deductive verification:

Using some logical formalism, prove formally that the software satisfies its specification.

Model checking:

Use some software to automatically check that the software satisfies its specification.

Testing:

Check executions of the software according to some coverage scheme.

Typical situation:

- Boss: Mark, I want that the new robot software will be flawless. OK?
- Mark: Hmmm. Well. Oh! Ah??? Where do I start?
- Bob: I have just the solution for you. It would solve everything.

Some concerns

- Which technique?
- Which tool?
- Which experts?
- What limitations?
- What methodology?
- At which points?
- How expensive?
- How many people?
- Needed expertise
- Kind of training
- Size limitations
- Exhaustiveness
- Reliability
- Evidence
- Expressiveness
- Support

Seven Myths of Formal Methods

- Myth 1: Formal methods can guarantee that software is perfect
- Myth 2: Formal methods are about program proving
- Myth 3: Formal methods are only useful for safety-critical systems
- Myth 4: Formal methods require highly trained mathematicians
- Myth 5: Formal methods increases the cost of development
- Myth 6: Formal methods are unacceptable to users
- Myth 7: Formal methods are not used on real, large-scale software

[Anthony Hall, IEEE Computer, Sep 1990]

Some exaggerations

Automatic verification can always find errors.

Deductive verification can show that the software is completely safe.

Testing is the only industrial practical method.

Advantages of Formal Methods

- Formal methods treat system components as mathematical objects and provide mathematical models to describe and predict the observable properties and behaviors of these objects.
- There are several advantages to using formal methods for the specification and analysis of real-time systems.
  - the early discovery of ambiguities, inconsistencies and incompleteness in informal requirements
  - the automatic or machine-assisted analysis of the correctness of specifications with respect to requirements
  - the evaluation of design alternatives without expensive prototyping

Our approach

Learn several methods (deductive verification, model checking, testing process algebra).

Learn advantages and limitations, in order to choose the right methods and tools.

Learn how to combine existing methods.

Things to do

- Check the kind of software to analyze.
- Choose methods and tools.
- Express system properties.
- Model the software.
- Apply methods.
- Obtain verification results.
- Analyze results.
- Identify errors.
- Suggest correction.

Different types of software

- Sequential.
- Concurrent.
- Distributed.
- Reactive.
- Real-time.
- Protocols.
- Abstract algorithms.
- Finite state.

Formal Specification Methods

- Logic
  - Z, VDM, First order logic, temporal logic
- State Machines
  - Finite state machines, communicating state machines, extended state machines
  - State Chart, Objectime, Automata, Timed Automat
- Petri Nets
- Process Algebra
  - CSP , CCS, ATP, ACSR
  - CWB, PARAGON

27

Verification Methods

- Verification is to show

  Behavior(Design) $\subseteq$ Behavior(Requirement)
- Verification Methods
  - Proof System : $SP \Rightarrow f$
  - Model Checking : $P_d \models f$
  - Behavioral Specification : P_d *sat* f

    E.g. 0 <=(# of coins in t) - (# of candies in t) <= 1
  - Algebraic(bisimulation, process containment):
    - $P_r \sim P_d$, $P_r \sqsubseteq P_d$,

28