IAF0530/IAF9530

Dependability and fault tolerance

Lecture 5
Testing Real-Time Systems

Gert Jervan
gert.jervan@pld.ttu.ee

Important

- Student presentations.
  - Presentation dates: 17/04, 24/04, 15/05, 22/05, 24/05, 01/06. Always at 12:00, IT-209. Register at the course homepage.
  - MSc students: 20 min
  - PhD students: 30 min
  - Mandatory to participate at least on three occasions (on top of your own presentation) and ask questions!
- Deadline of the final report:
  - 01/06 for those who are making presentation 17/04-22/05
  - 05/06 for those who are making presentation 24/05-01/06

Remarks by Bill Gates
17th Annual ACM Conference on Object-Oriented Programming, Seattle, Washington, November 8, 2002

"… When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development. We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time…

… We've probably changed the industry we're in. We're not in the software industry; we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing."

Remarks by Bill Gates (cont.)

"…The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one."

"… Well, one of the interesting questions is, when you change a program, … what portion of these test cases do you need to run?"

Testing Real-Time Systems

Distributed
Self-Checking

System Testing

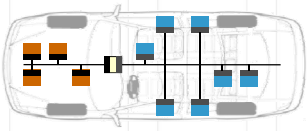
HW Testing

SW Testing

HW/SW Testing
(system testing)

Real-Time Systems

- *Real-Time System* – system, which is required to adhere not only functional but also tempoal requirements ("timing constraints" or "deadlines")
- RT-systems:
  - Hard RT-systems
  - Soft RT-systems



7

Real-Time Systems Testing

- Inherits issues from concurrent systems
  - Problems becomes harder due to time-constraints
    - More sensitive to probe-effects
    - Timing/order of inputs become more significant
- Adds new potential problems
  - New failure types
    - E.g. Missed deadlines, Too early responses…
  - Test inputs → Execution times
  - Faults in real-time scheduling
    - Algorithm implementation errors
    - Assumption about system wrong

8

Real-Time Systems Testing

- Pure time-triggered systems
  - Deterministic
  - Test-methods for sequential software usually apply
- Fixed priority scheduling
  - Non-deterministic
    - Limited set of possible execution orders
  - Worst-case w.r.t timeliness can be found from analysis
- Dynamic (online) scheduled systems
  - Non-deterministic
    - Large set of possible execution orders
  - Timeliness needs to be tested

9

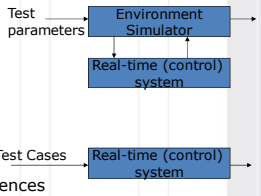
Testing Timeliness

- Aim : Verification of specified deadlines for individual tasks
  - Test if assumptions about system hold
    - E.g. worst-case execution time estimates, overheads, context switch times, hardware acceleration efficency, I/O latency, blocking times, dependency-assumptions
  - Test system temporal behavior under stress
    - E.g. Unexpected job requests, overload management, component failure, admission control scheme
- Identification of potential worst-case execution orders
- Controllability needed to test worst-case situations efficiently

10

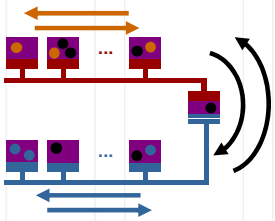
Testing Embedded Systems

- System-level testing differs
  - Performed on target platform to keep timing
- Closed-loop testing
  - Test-cases consist of parameters sent to the environment simulator
- Open-loop testing
  - Test-cases contain sequences of events that the system should be able to handle



11

Distributed Real-Time Systems



- Distributed applications
  - On a single cluster
  - On several clusters
- Motivation
  - Reduce costs: use resources efficiently
  - Requirements: close to sensors/ actuators
- Distributed applications are difficult to…
  - Analyze (e.g., guaranteeing timing constraints)
  - Design (e.g., efficient implementation)

12

Testing Distributed RT-Systems

- Problems with distributed systems:
  - Increased complexity
  - The difficulties of observing and monitoring
  - Non-reproducible behaviour of the system
  - The lack of synchronized global clock and, consequently, the difficulties of unambiguously defining a "global state"

13

Testing Distributed RT-Systems

- Observability
  - What?
  - How?
  - When?
- Controllability
- Auxiliary outputs, interactive debuggers

14

Observability Issues

- Probe effect (Gait,1985)
  - "Heisenbergs's principle" - for computer systems
  - Common "solutions"
    - Compensate
    - Leave probes in system
    - Ignore
- Must observe execution orders
  - Gain coverage

15

Controllability Issues

- To be able to test correctness of a particular execution order we need control
  - Input data to all tasks
    - Initial state of shared data/buffers
  - Scheduling decisions
    - Order synchronization/communication between tasks

16

Testing Distributed RT-Systems

- **Reproducibility**
  - *Regression testing* – retesting after errors have been corrected
    - errors truely corrected
    - no new errors
  - A distributed system may be non-reproducible due to nondeteminism in it's hardware, software or operating system

17

Testing Distributed RT-Systems

- **Obtaining reproducibility**
  - Language-based approach
    - Enforcing the identified scenarios during execution
    - All solutions rely on source code transformations
  - Implementation based approach
    - Collecting all missing information during an execution of the system
    - Event histories or traces

18

Testing Distributed RT-Systems

- Disadvantages of implementation based approach:
  - Special dedicated HW (to monitor)
  - Large amount of information
  - Can we guarantee the correctnes of reply?
  - Modified programs. What happens with event histories. Are they still valid?
  - Event histories can be used only on target systems

19

Testing Distributed RT-Systems

- Interdependence of Obsevability and Reproducibility
  - Not independent!
  - Probe effect

20

Testing Distributed RT-Systems

- **The host/target approach**
  - Host - development
  - Target - execution
- Testing on the host system is used for (functional) unit testing and preliminary integration testing (as much as possible)
- Testing on the target system involves completing the integration test and performing the system test. Also performance, timing, etc.

21

Testing Distributed RT-Systems

- Environment simulation (for target system test)
  - Simulated v. real environment:
    - Safety and/or cost considerations.
    - "rare event" situations
    - More control over simulated environment
    - Easier to obtain responses and test results
  - On-line v. off-line test data generation:
    - Need to generate large amounts of input data
    - Runs cost-effectively

22

Testing Distributed RT-Systems

- Representativity
  - Only small number of real-world scenarios can be anticipated and taken into account.
  - Only a fraction of those anticipated real-world scenarios can be tested due to the combinatorial explosion of possible event and input combinations.
- Test coverage - how many of the anticipated real-time scenarios can be or have been covered by corresponding test scenarios.

23

Self-checking distributed systems

- Run-time checking of the effects of faults on system behaviors needs to be carried out continuously.
- Reliability – the key to distributed SW quality

24

Self-checking distributed systems

- Aspects to design correct SW:
  - Reliability with which the SW specifications are adequately described and correctly implemented in the actual implementation.
  - Run-time checking

25

Self-checking distributed systems

- Fault-secure systems are systems, where faults may be enforced not to propagate.
  - Faults are not visible or have no effect
  - Faults are visible, but it's easy to notice that an error exists
- Self-testing – System is self testing when there exists testing behavior, occurring during the run-time behavior of the system, such that this fault will be propagated to the output and it's easy to notice, that there is a fault (out of predefined set of values)
- System is self-checking for a set of faults, if whatever a fault belonging to this set, it is fault-secure and self-testing.

26

Self-checking distributed systems

- Worker-observer
  - the worker is a classical implementation of the system behavior
  - the observer is a given redundant implementation whose outputs are comparable with the outputs of the worker.
- To obtain observing behavior:
  - Redundancy
  - Reference
  - Visibility
    - Worker cooperates with the observer
    - Worker behavior can be spied by the observer

27

Self-checking distributed systems

- A formal observer is a subsystem designed to check distributed behaviors where:
  - Its software is independent of the specific protocols to be checked in the considered system;
  - Its data are defined by the protocols to be checked and this data can be formally specified and verified.

28

Self-checking distributed systems

- Design of the system
  - write a description of the beavior of the system to be implemented;
  - Implement the system itself, i.e., the worker;
  - From the description of the worker, select (based on experience) that part of the behavior which should be observed and write a formal model of it.

29

Self-checking distributed systems

- The system is quasi self-checking if
  - It is an observer-worker system
  - The observer is a formal observer.
- For "real-life" only part of the system will be modelled.
- Formal model must be able to
  - Express simplified specifications of distributed systems
  - Support verification procedures
  - Be able to act as a basis for implementing the observer.

30

Few testing criteria exists for concurrent systems

- Number of execution orders grow exponentially with # synchronization primitives in tasks
  - Testing criteria needed to bound and selecting subset of execution orders for testing
- E.g. Branch / Statement coverage not sufficient for concurrent software
  - Still useful on serializations
  - Execution paths may require specific behavior from other tasks
- Data-flow based testing criteria has been adapted
  - E.g. define-use pairs

31

Determinism vs. Non-Determinism

- Deterministic systems
  - Controllability is high
    - input (sequence) suffice
  - Coverage can be claimed after single test execution with inputs
  - E.g. Filters, Pure "table-driven" real-time systems
- Non-Deterministic systems
  - Controllability is generally low
  - Statistical methods needed in combination with input coverage
  - E.g.
    - Systems that use random heuristics
    - Behavior depends on execution times / race conditions

32

Test execution in concurrent systems

- Non-deterministic testing
  - "Run, Run, Run and Pray"
- Deterministic testing
  - Select a particular execution order and force it
  - E.g. Instrument with extra synchronizations primitives
    - (No timing constraints make this possible)
- Prefix-based Testing (and Replay)
  - Deterministically run system to a specific (prefix) point
  - Start non-deterministic testing at that specific point

33

Questions?