Dependable Real-Time Systems

Lecture #2

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Dependability in real-time systems

Dependability ...

"... is that property of a computer system which allows reliance to be justifiably placed on the service it delivers ..."

IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance

"... includes as special cases the notions of reliability and safety ..."

Jean-Claude Laprie

Dependability in real-time systems

Components of dependability: [Laprie]

- Attributes:
  - The ways and measures by which the quality of a dependable service can be appraised
  - Reliability, Availability, Safety (+ Confidentiality, Integrity, Maintainability)
- Threats:
  - Circumstances causing or resulting in non-dependability
  - Faults, Errors, Failures
- Means:
  - The methods, tools and solutions required to deliver a dependable service with the required confidence
  - Fault prevention, Fault removal, Fault tolerance

Dependability in real-time systems

Attributes of dependability:

- Reliability:
  - The ability of a system to continuously deliver the service that is expected (according to specifications).
- Availability:
  - The ability of a system to deliver its expected service at the time instant the service is requested.
- Safety:
  - The ability of a system to guarantee that an incident does not occur, whether or not the system delivers its expected service.

    Incident: unplanned event or series of events that may result in death, injury, occupational illness, damage to (or loss of) equipment (or property), or environmental harm.

Dependability in real-time systems

Attributes of dependability: (in terms of probability)

- Reliability:
  - The probability that the system is still functioning at time t, given that it was functioning at time t = 0.
- Availability:
  - The probability that the system is functioning at the time instant its service is requested.
- Safety:
  - The probability that the system is either functioning, or are in a safe (non-functioning) state.

Dependability in real-time systems

Attributes of dependability: (in terms of probability)

- Reliability:
  - The probability that the system is still functioning at time t, given that it was functioning at time t = 0.
  - Expected life time: MTTF = 1 / λ [hours] (mean time to failure)
    λ : fault intensity [faults / hour] (typically, faults per million hours)
- Availability:
  - The probability that the system is functioning at the time instant its service is requested.
  - Expected repair time: MTTR = 1 / μ [hours] (mean time to repair)
    μ : repair intensity [repairs / hour]
  - Availability = MTTF / (MTTF + MTTR)

Dependability in real-time systems

Threats against dependability:

- Fault:
  - The system has encountered a condition that <u>may</u> lead to an undesired system state (error).
  - The condition may be permanent, transient or intermittent.
- Error:
  - The system has changed into an undesired state that <u>may</u> lead to an inability to deliver expected service (failure).
- Failure:
  - The system does not deliver the service that is expected, according to specification.
  - The failure can be in the time domain or value domain.

Dependability in real-time systems

What causes faults?

- Design faults:
  - Incomplete specifications
  - Erroneous system models (e.g., underestimated WCET)
  - Insufficient formal checking (e.g., no schedulability analysis)
- Component defects:
  - Manufacturing effects (in hardware or software)
  - Wear and tear due to component use (“aging”)
- Environmental effects:
  - High stress (temperature, G-forces, vibrations)
  - Electromagnetic or elementary-particle radiation

Dependability in real-time systems

What types of (hardware) faults are there?

- Permanent faults:
  - Total failure of a component
  - Remains until component is repaired or replaced
  - Caused by, e.g., short circuit, broken wire, depleted battery
- Transient faults:
  - Temporary malfunction of a component
  - Caused by, e.g., radiation, power fluctuations
- Intermittent faults:
  - Repeated occurrences of transient faults
  - Caused by, e.g., overheating, loose wires

Dependability in real-time systems

What types of (software) faults are there?

- Permanent faults:
  - Total failure of software code
  - Remains until software is repaired or restarted
  - Caused by, e.g., corrupt data structures, unterminated tasks
- Transient faults:
  - Temporary malfunction of software code
  - Caused by, e.g., data-dependent bugs in the code
- Intermittent faults:
  - Repeated occurrences of transient faults
  - Caused by, e.g., memory-pointer problems

Dependability in real-time systems

Examples of fault → error → failure chains:

- Scenario #1:
  1. A temporary bit flip occurs in a processor register due to elementary-particle radiation (hardware fault)
  2. The bit flip occurs in a register which is being used as a loop counter, leading to a too high number of loop iterations in a high-priority task, thereby causing the real execution time of the task to exceed the predicted WCET (error)
  3. The additional interference by the high-priority task leads to a deadline miss in a lower-priority task which in turn negatively affects system behavior (failure)

Dependability in real-time systems

Examples of fault → error → failure chains:

- Scenario #2:
  1. A piece of software code contains an unintentional bug that may try to store a string of n+1 characters into a buffer that only has reserved space for n characters (software fault)
  2. A string store operation involving n+1 characters has been done to the under-dimensioned buffer, causing the last stored character to overwrite the value of another variable (error)
  3. The variable whose value was overwritten is of integer type, and its new value changes the program behavior in such a way that a set of CAN messages are transmitted more frequently than stated in the system specification (failure)

Dependability in real-time systems

Examples of fault → error → failure chains:

- Scenario #2b: (in another system on the same CAN bus)
  1. A burst of messages occurs on the CAN bus, causing the inter-arrival time of two subsequent CAN messages to be shorter than assumed in the specification (design fault)
  2. The unexpectedly-short message inter-arrival time causes the CAN interrupt handler (a high-priority task) to execute more frequently than predicted (error)
  3. The additional interference by the interrupt handler task causes a lower-priority task to miss its deadline, which in turn negatively affects system behavior (failure)

Thus, a failure of one system can cause a fault in another system!

Dependability in real-time systems

Examples of fault → error → failure chains:

- These examples all led to time failures.

  Failures in the time domain means that service is delivered in an untimely manner with respect to the system specification.

  Fail late/early: the system produces correct services in the value domain, but may suffer from a 'late' or 'early' timing error.

  Fail silent: the system produces correct services until it suddenly produces no service at all (a k a "omission failure")

- Give examples that can lead to value failures.

  Failures in the value domain means that some data resulting from the service is wrong with respect to the system specification.

Dependability in real-time systems

Means for dependability:

- Fault prevention:
  - Introduction of faults in the system is avoided by means of reliable hardware and software components, and formal methods for verifying the correctness of these components.
- Fault removal:
  - Existing faults are identified by testing and removed by repair.
- Fault tolerance:
  - Faults that occur (despite any preventive measures) while the system is in mission are handled by means of component redundancy, implemented as fault masking or error detection.

Dependability in real-time systems

Fault prevention:

- High-quality hardware:
  - Military specified components that can withstand extra abuse from the environment (MIL-SPEC), and physical screening to protect components from radiation.
- High-quality software:
  - Programming languages with strong type checking, data abstraction, and modularity, and software engineering methods with revision control, etc.
- Formal methods:
  - Schedulability analysis to prove timing correctness, and model checkers to prove absence of deadlock, in concurrent software.

Dependability in real-time systems

Fault tolerance:

- Full fault tolerance: (no fail)

  Fault masking: uses static redundancy with active backup components; completely transparent to the system.

  Error detection: uses dynamic redundancy with passive backup components; requires the system to take suitable action.

- Graceful degradation: (fail soft)
  - Combines error detection with a subsequent action that takes the system into a service mode with guarantees of lower quality.
- Controlled shutdown: (fail safe)
  - Combines error detection with a subsequent action that takes the system into a (non-functioning) safe state.

Fault-tolerant techniques

Detecting timing errors during task execution (no fail):

- Caused by:
  - Overrun: inter-arrival times shorter than assumed (sporadic overrun) or execution time exceeds WCET (execution overrun)
  - Silent failure: a task suddenly stops delivering service
- Detect with:
  - Watchdog: time-out mechanism that monitors the temporal behavior of tasks, and compares against expected behavior.
- Action taken:
  - Allow: accept the overrun (if system service semantics allow)
  - Terminate: do not accept the overrun (if semantics allow)
  - Reconfigure: let other task replace the lost one (backup task)

Fault-tolerant techniques

Masking timing errors during task execution (no fail):

- Caused by:
  - Sporadic overrun: real inter-arrival times shorter than assumed
- Detect with:
  - Watchdog: time-out mechanism that monitors the temporal behavior of tasks, and compares against expected behavior.
- Action taken:
  - Regulate: adjust inter-arrival times (restore intended behavior)

Fault-tolerant techniques

Detecting value errors during task execution (no fail):

- Caused by:
  - Hardware or software faults: bit flips in registers, bugs in code
- Detect with:
  - Duplex components: compare results between two identical calculations executed by redundant hardware or software.
  - Constraint check: compare result against known value bounds.
- Action taken:
  - Dispose: do not use value (if system service semantics allow)
  - Replace: with a reasonable value (if semantics allow)
  - Re-execute calculation (if system service time allows)

Fault-tolerant techniques

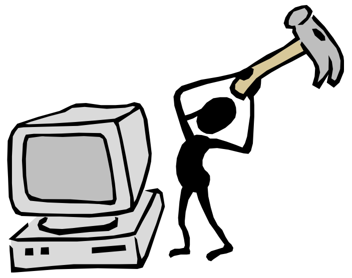
Masking value errors during task execution (no fail):

- Caused by:
  - Hardware or software faults: bit flips in registers, bugs in code
- Masking (correction) with:
  - Voting: compare results between three identical calculations executed by redundant hardware or software and pick the value that dominates (majority voting).
  - Error correction: restore to original value (using, e.g., error correcting codes).
- Action taken:
  - None: masking is transparent to system!

Fault-tolerant scheduling

To extend real-time computing towards fault-tolerance, the following issues must be considered:

1. What is the fault model used?
   - What type of fault is assumed?
   - How and when are errors detected?
2. How should fault-tolerance be implemented?
   - Using temporal redundancy (re-execution)
   - Using spatial redundancy (replicated tasks/processors)
3. What scheduling policy should be used?
   - Extend existing policies (cyclic executive / priority scheduler)
   - Suggest new policies …



Fault-tolerant scheduling

What fault model is used?

Type of fault:

- Transient, intermittent and/or permanent faults
- For transient or intermittent faults: is there a minimum inter-arrival time between two subsequent faults?

Error detection:

- Comparison (after task execution)
- Constraint checking (during task execution)
- Watchdogs (during task execution)

Fault-tolerant scheduling

How is the fault-tolerance implemented?

Temporal redundancy:

- Tasks are re-executed to provide replicas for voting decisions
- Tasks are re-executed to recover from a fault
- Re-execution may be from beginning or from check-point
- Re-executed task may be original or simplified version

Spatial redundancy:

- Replicas of tasks are distributed on multiple processors
- Identical implementations or different implementations (N-version programming) of tasks
- Voting decisions are made to detect errors or mask faults

Fault-tolerant scheduling

How do we make existing techniques fault-tolerant?

Single-processor scheduling:

- Use a state-of-the-art scheduler and use any surplus capacity (slack) to re-execute tasks that experience errors during their execution.
- Alt #1: The slack is reserved *a priori* and can be accounted for in a schedulability test. This allows for hard real-time guarantees (under the assumed fault model)
- Alt #2: re-executions can be modeled as aperiodic tasks. The slack is then extracted dynamically at run-time by dedicated aperiodic servers. This allows for statistical guarantees.

Fault-tolerant scheduling

How do we make existing techniques fault-tolerant?

Multiprocessor scheduling:

- Generate a multiprocessor schedule that includes primary and backup (active or passive) tasks.
- Execute the primary tasks in the normal course of things.
- Active backup tasks always execute in parallel with the primary.
- Passive backup tasks are only activated if the primary fails.
- Trick #1: Let a passive backup handle multiple primaries in the schedule (overloading).
- Trick #2: De-allocate resources reserved for a passive backup if its primary completes successfully.