Co-funded by the European Union
Erasmus+
AGENCE ERASMUS+ FRANCE · EDUCATION FORMATION
La Région Auvergne-Rhône-Alpes

Real-time safety and fault injection

Material prepared by A. Bakke, B. Thapa, H. Hassan and revised by Stefano Di Carlo, Alessandro Savino, Alessio Carpegna and Cristiano Chenet

Outline

Introduction to real-time safety

Fault injection

Fault prevention and Fault tolerance

Techniques

Summary

Introduction to real-time safety

Safety critical real-time systems:

- Applications characterized by the high risk involved, therefore Reliability and safety requirements are usually stringent in them.
- Hard real-time systems, missing a deadline is not acceptable.
- Failures lead to catastrophic consequences may cost lives or cause large financial losses.

Suggested reading: Hilburn & Zalewski

Introduction to real-time safety

Examples of safety-critical systems:

- **Aerospace Industries:** Extremely complex embedded systems with critical safety specifications.

  An example of these systems is “A digital flight control system”.

  - ❏ A system that interprets the pilot's control inputs and sends appropriate commands to the control surfaces and engines.

  - ❏ The criticality of this system is to provide a reliability to meet the certification requirement of the probability of catastrophic failure rate to be less than 10^-9 /hr.

Introduction to real-time safety

- **Nuclear Industries:** Include critical sensors and control systems for functioning and monitoring the nuclear plants, preventing the reactor operation outside of normal conditions which may result in a core meltdown and an emission of radiation.

The most safety-critical sensors are those measure temperature, pressure, coolant level and flow. Such sensors must operate sufficiently fast in harsh environments with high sensitivity

An example of these sensors is “resistance temperature detectors (RTDs)”.

- ❏ key components in nuclear reactor safety systems, are expected to provide 0.1% accuracy and respond to a temperature change in less than 4 seconds.

Introduction to real-time safety

Embedded system failure Sources:

Four sources of faults which can cause a system failure:

- Inadequate specification, including faults that stem from misunderstanding the interactions between the program and the environment.
- Design errors in software.
- Failure of one or more hardware components. (ex: hard error and soft error)
- Interference or failure in the communication subsystem.

Introduction to real-time safety

Soft Error: one of the possible sources of system failures, it is an error occurrence in a computer's memory system that changes an instruction in a program or a data value.

It will not damage a system's hardware (can be remedied by a reset)
The only damage is to the data that is being processed, which is not acceptable in real-time systems.

Cause:

Data being processed is hit with a noise phenomenon, typically when the data is on a data bus. The computer interpret the noise as a data bit, which can cause errors in addressing or processing program code, or saved in memory causing failure when processed.

Outline

Introduction to real-time safety

Fault injection

Fault prevention and Fault tolerance

Techniques

Summary

Fault Injection

A validation technique of the systems reliability, to ensure it can withstand and recover from error conditions.
It evaluates the correct behavior of software with faults that are injected intentionally.

Techniques:

- Hardware-based fault injection
- Software-based fault injection
- Simulation-based fault injection
- Emulation-based fault injection

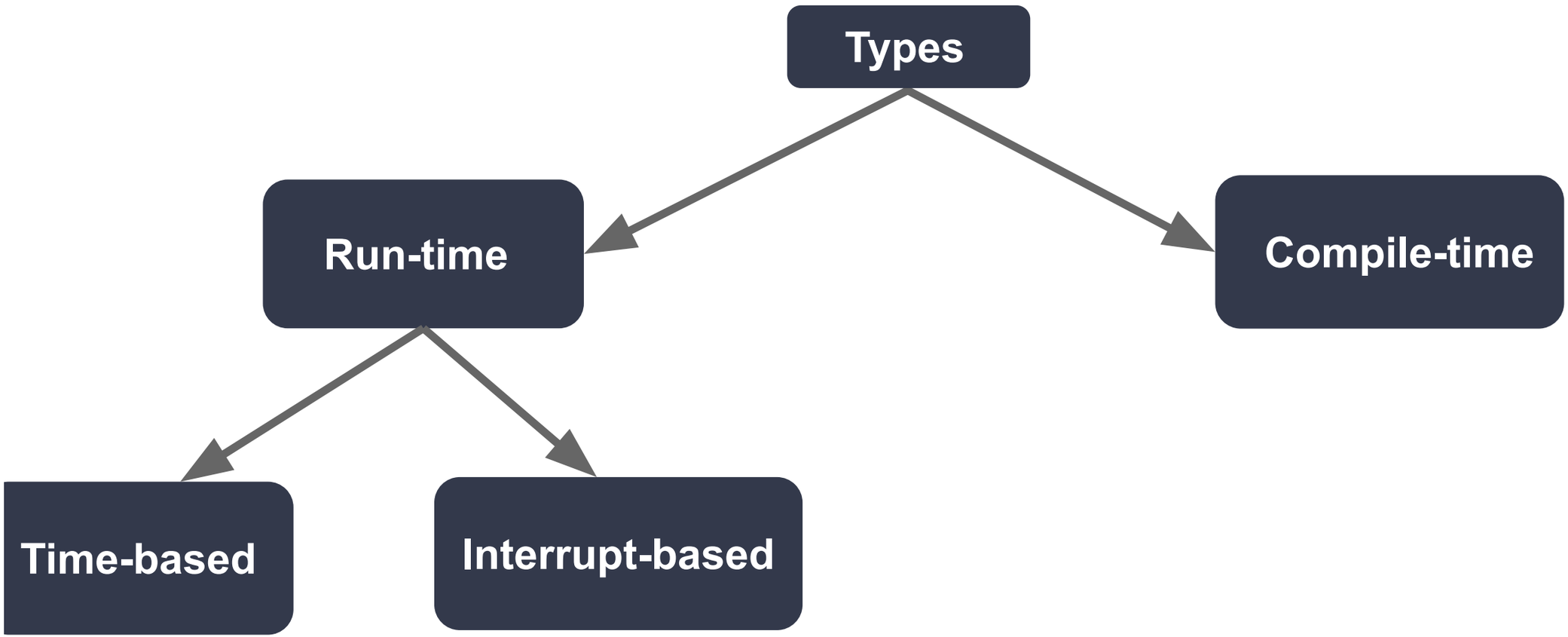
ıggested reading: Ziade, Ayoubi & Velazco

Fault Injection

Software-based fault injection

Software fault injections are more oriented towards implementation details, and can address program state as well as communication and interactions. Faults are mis-timings, missing messages, replays, corrupted memory or registers, faulty disk reads.

Software-based Fault Injection



Software-based Fault Injection

Tools Example

FERRARI (Fault and Error Automatic Real-Time Injection)

- Developed at the University of Texas at Austin.
- Uses software traps to inject CPU, memory, and bus faults.
- The traps are triggered either by the program counter or by a timer. When the traps are triggered, the trap handling routines inject faults.
- Inject faults at specific fault locations, by changing the content of selected registers or memory locations to emulate actual data corruptions.The faults injected result in an address line error, a data line error, and a condition bit error.

Outline

Introduction to real-time safety

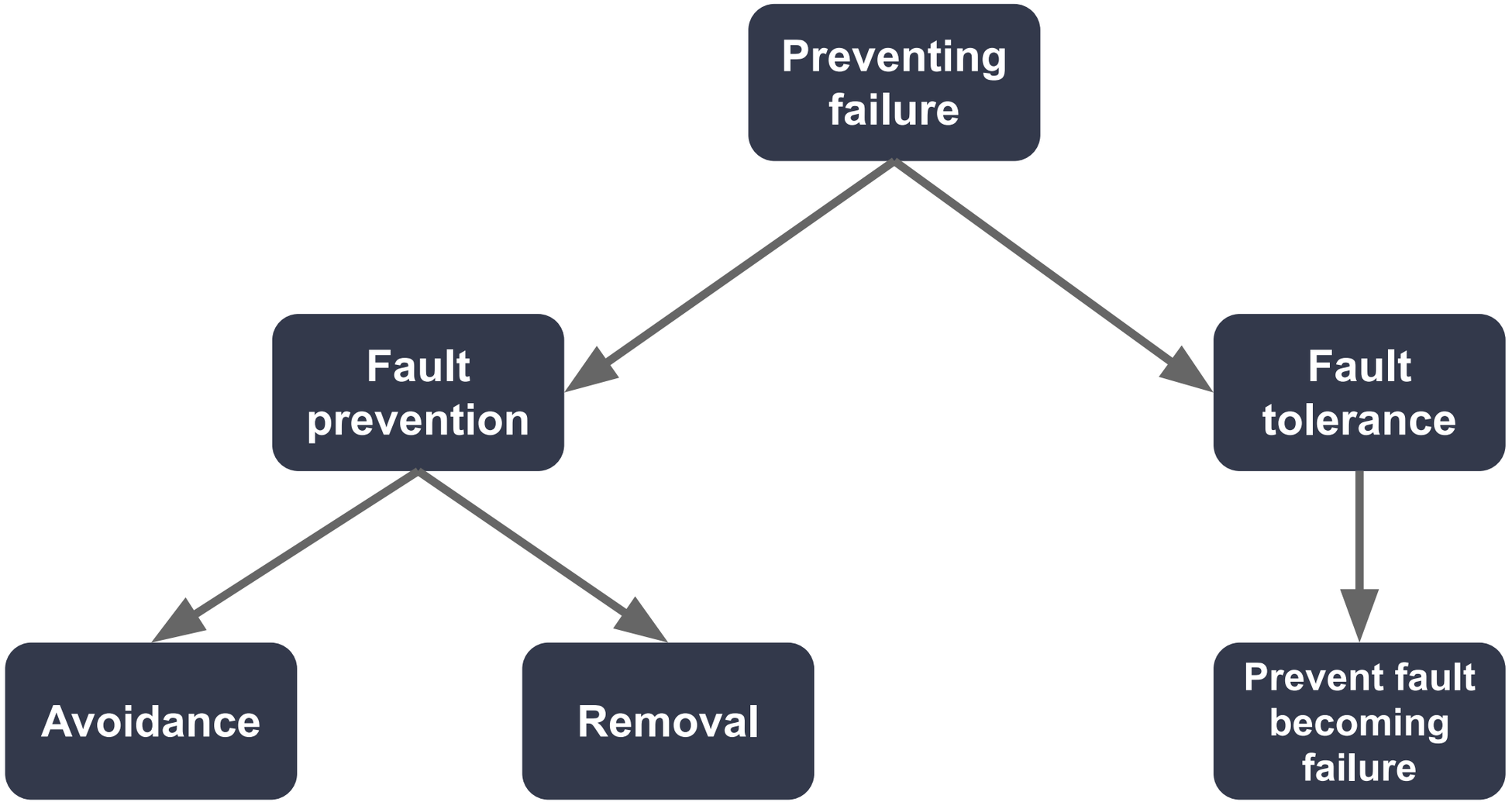
Fault injection

Fault prevention and Fault tolerance

Techniques

Summary


Fault
Error
Failure


Preventing failure
Fault prevention
Fault tolerance
Avoidance
Removal
Prevent fault becoming failure

Fault prevention

Avoidance: Programming methodologies and techniques for better, maintainable code.
Suggested reading: “Code complete” by Steve McConnel

Removal: Testing code and fixing errors. Inspecting code.

Fault tolerance

Burns & Wellings (Ch2):

Full fault tolerance: The system continues to operate in the presence of faults, albeit for a limited period, with no significant loss of functionality or performance.

Graceful degradation: The system continues to operate in the presence of faults, accepting a partial degradation of functionality or performance during recovery or repair.

Fail safe: The system maintains its integrity while accepting a temporary halt in its operation.

Outline

Introduction to real-time safety

Fault injection

Fault prevention and Fault tolerance

Techniques

Summary

Fault tolerance

Acceptance tests: Test for acceptable results instead of faults

Redundancy: N-version programming

Suggested reading: Burns & Wellings (Ch2), Gray & Reuter (Ch 3.7)

Acceptance tests: Detection

Burns & Wellings (Ch2):

Replication check

Timing check

Reversal check

Coding check

Reasonableness check

Structure check

Dynamic reasonableness check

Example: Timing test (monitoring)

Idea: Profile WCET and monitor execution time of the task.

Profiled (estimated) WCET:

- Requires knowledge of the system
- Should be larger or equal to the real WCET

Monitoring:

1. Watchdog timer
   a. Reset a timer at the end of every task execution.
   b. Timer triggers another task which handles the fault
2. Underlying run-time support system
   a. Example: The Real-time specification for Java has an built-in support system for monitoring and handling deadlines

Note: We do NOT ensure that the task is working correctly, only that it functions on time

Acceptance tests: Recovery

Forward: Try to continue from the current state by making corrections to the system state

Indicative of full fault tolerance or graceful degradation

Backward: Restore system to an older safe(er) state

Indicative of a fail safe fault tolerant system.

Suggested reading: Burns & Wellings (Ch2), Gray & Reuter (Ch 3.7)

Outline

Introduction to real-time safety

Fault injection

Fault prevention and Fault tolerance

Techniques

Summary

Summary

A **safety critical system** is often characterized as a hard real-time system, a missed deadline is non acceptable.

A **soft error** is a source of system failures

Fault injection validates a systems reliability by injecting faults intentionally.

Fault tolerance allows the system to function with **soft errors** present.

Error-detection can be done using numerous methods depending on the system.

Error-recovery is done **forward** or **backward**, often implying different levels of fault tolerance

Source material collected

“Real-Time Systems and Programming Languages (Fourth Edition)" by Alan Burns and Andy Wellings, Chapter 2.

Topics: Fault tolerance levels, error detection, failure modes, redundancy, fault prevention

“Transaction Processing” by Jim Gray and Andreas Reuter, Chapter 3.7

Topics: Process pairs, redundancy, recovery

“Code complete” by Steve McConnel, Chapters 6, 7, 11, and 31.

Topics: Coding principles and checklists to increase software quality and safety

“A Flexible WCET Analysis Method for Safety-Critical Real-Time System using UML-MARTE Model Checker” by Ning Ge, Marc Pantel, Bernard Berthomieu.

Topics: Profiling WCET

“*REAL-TIME SAFETY-CRITICAL SYSTEMS: AN OVERVIEW.” by* Thomas Hilburn & Janusz Zalewski:

Topics: Safety critical systems and real-time safety

“ *A Survey on Fault Injection Techniques” by* Haissam Ziade , Rafic Ayoubi , and Raoul Velazco (2004)

Topics: Fault injections