CprE 558: Real-Time Systems

Lectures 15-16:

Dependability Concepts

& Faul-Tolerance

Dependable System

- A system is ***dependable*** when it is trustworthy enough that reliance can be placed on the service that it delivers. For a system to be dependable, it must be
  - **Available** - e.g., ready for use when we need it.
  - **Reliable** - e.g., able to provide continuity of service while we are using it.
  - **Safe** - e.g., does not have a catastrophic consequence on the environment.
  - **Secure** - e.g., able to preserve confidentiality.

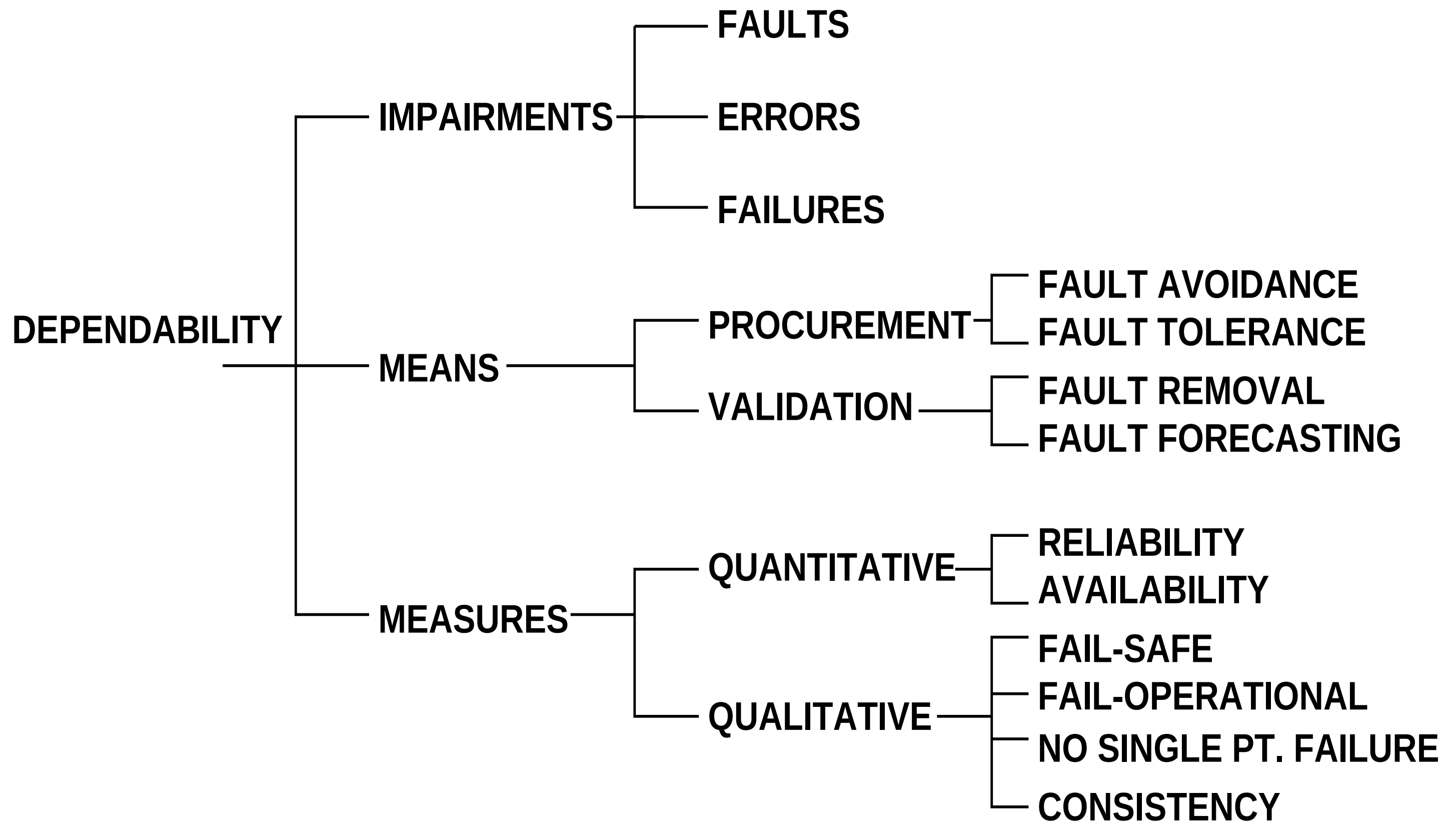
Why Dependability ?

- With a greater reliance on computers in a variety of safety-critical applications, the consequences of failure and down time have become more severe.
- For example, in safety-critical applications - such as flight control, medical life support, process control, telecommunication switching, and on-line transaction processing systems - failure of computing resources can cost lives and/or money.

Example for Dependable Systems

- The reliability figure usually stated as a goal for computer systems in commercial aircraft is less than 10^{-9} failures per hour.
- Modern telephone switching systems achieve a down time of at most one hour in 40 years.
- Medical life support system.
- Command and control systems.
- Process control applications.

Attributes of Dependability



Approaches to Achieving Dependability

- **Fault Avoidance** - how to prevent, *by construction*, the fault occurrence or introduction.
- **Fault Removal** - how to minimize, *by verification*, the presence of faults.
- **Fault Tolerance** - how to provide, *by redundancy*, a service complying with the specification in spite of faults.
- **Fault Forecasting** - how to estimate, *by evaluation*, the presence, the creation, and the consequence of faults.

Fault Avoidance

- Fault avoidance uses various tools and techniques to design the system in such a manner that the introduction of faults is minimized.
- A fault avoided is one that does not have to be dealt with at a later time.
- Techniques used include design methodologies, verification and validation methodologies, modeling, and code inspections and walk-throughs.

Fault Removal

- Fault Removal uses verification and testing techniques to locate faults enabling the necessary changes to be made to the system.
- The techniques include unit testing and integration testing.
- It is generally much more expensive to remove a fault than to avoid a fault.

Fault Tolerance

- A system built with fault tolerance capabilities will manage to keep operating, perhaps at a *degraded level*, in the presence of these faults.
- In other words, fault-tolerance is informally defined as the ability of a system to deliver the *expected* service even in the presence of faults.
- For a system to be fault-tolerant, it must be able to detect, diagnose, confine, mask, compensate and recover from faults.

Fault Forecasting

- It is possible to observe the behavior of a system and use this information to take action to compensate for faults before they occur.
- When a system deviates from its normal behavior, even if the behavior continues to meet system specifications, it may be appropriate to reconfigure the system to reduce the stress on a component with a high failure potential.

Achieving Dependability - Summary

- Fault avoidance and fault tolerance may be seen as constituting dependability **procurement:** how to *provide* the system with the ability to deliver the specified service.
- Fault removal and fault forecasting may be seen as constituting dependability **validation:** how to *reach confidence* in the system's ability to deliver the specified service.

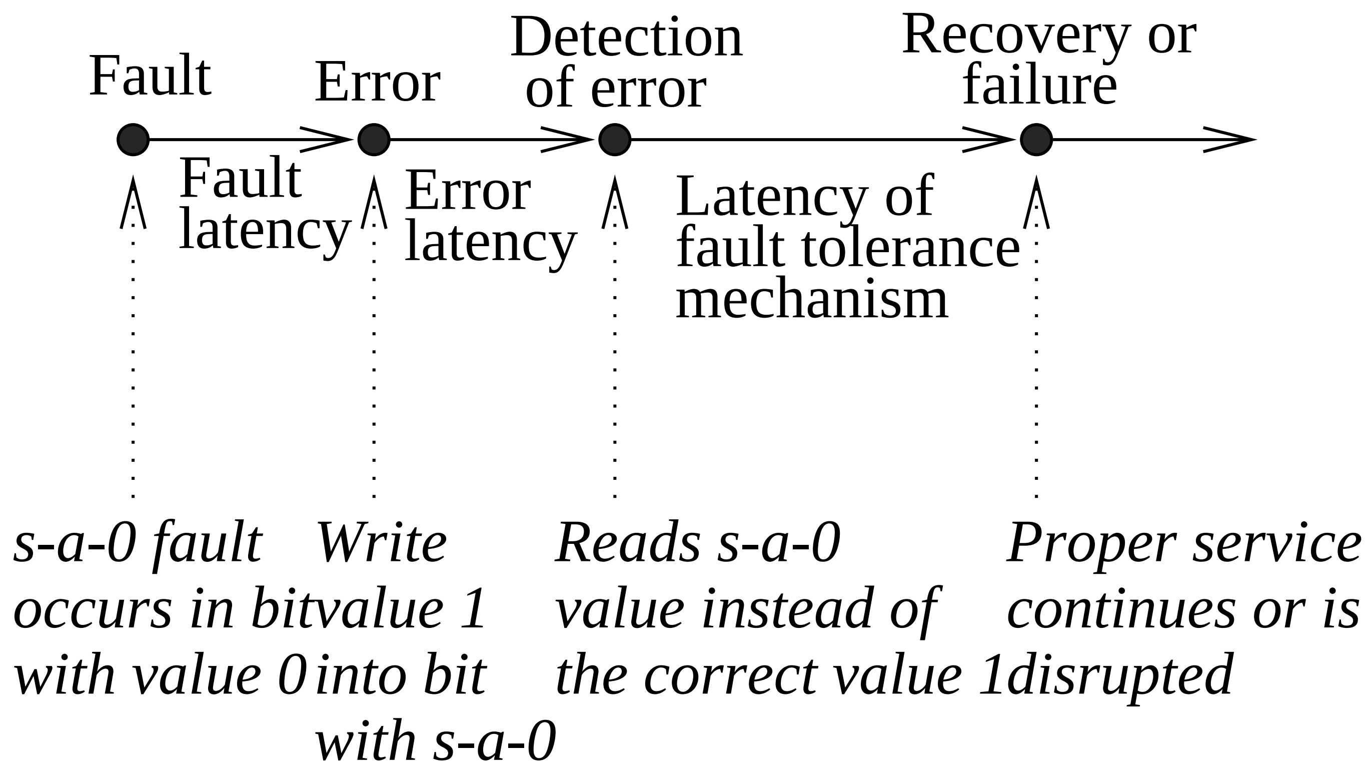
Fault, Error, and Failure

- A **fault** is a deviation in a hardware or software component from its intended function.
- An **error** is a manifestation of a fault in a system, in which the logical state of an element differs from its intended value.
- The time between fault occurrence and the first appearance of an error is called the **fault latency**.
- The time between occurrence of an error and its detection is called **error latency**.

Fault, Error, and Failure (Contd.)

- When the fault-tolerance mechanisms **detect** an error, they may initiate several actions to handle the fault and contain its errors.
- Recovery occurs if these actions are successful; otherwise, the system eventually malfunctions and a **failure** occurs.

Example of a fault, an error, and a failure


Fault
Error
Detection of error
Recovery or failure
Fault latency
Error latency
Latency of fault tolerance mechanism
s-a-0 fault occurs in bit with value 0
Write value 1 into bit with s-a-0
Reads s-a-0 value instead of the correct value 1
Proper service continues or is disrupted

Fault

- Faults can arise during all stages in a computer system's evolution - specification, design, development, manufacturing, assembly, and installation - and throughout its operational life.
- Most faults that occur before full system deployment are discovered through testing and eliminated.
- Faults that are not removed can reduce a system's dependability when it is in the field.
- A fault can be classified by its duration, nature of output, and correlation to other faults.

Fault types - Based on Duration

- **Permanent faults** are caused by irreversible device failures within a component due to damage, fatigue, or improper manufacturing. Once a permanent fault has occurred, the faulty component can be restored by replacement or repair.
- **Transient faults** are triggered by environmental disturbances such as voltage fluctuations, electro-magnetic interference, or radiation. These events typically have a short duration, returning the affected circuitry to a normal operating state without causing any lasting damage.

Fault types - Based on Duration (Contd.)

- **Intermittent faults** are those which tend to oscillate between periods of erroneous activity and dormancy, may also surface during system operation. They are often attributed to design errors that result in marginal or unstable hardware.
  Example: fault due to a loose wire.

Fault Types - Based on Nature of Output

- **Malicious fault:** Whenever a fault can cause a unit to behave arbitrarily, malicious or **Byzantine failure** is said to happen.
  - A sensor sending conflicting outputs to different processors.
  - An output line that stays afloat rather than stuck-at to 0 or 1 is considered to be malicious, because it is difficult to conclude consistently whether the output is 0 or 1.
- **Non-malicious:** Stuck-at faults are non-malicious.

Malicious faults are much harder to detect than non-malicious faults.

Fail-stop Unit

- A unit is said to be *fail-stop* if it responds to up to a certain maximum number of faults by simply stopping, rather than producing incorrect output.
- A fail-stop unit typically has many processors running the same tasks and comparing the outputs. If the outputs do not agree, the whole unit turns itself off.
- A system is said to be *fail-safe* if one or more safe states can be identified, that can be accessed in case of a system failure, in order to avoid catastrophe. *Example:* Railway signaling.

Fault Types - Based on Correlation

- Components fault may be independent of one another or correlated.
- A fault is said to be **independent** if it does not directly or indirectly cause another fault.
- Faults are said to be **correlated** if they are related. Faults could be correlated due to physical or electrical coupling of units.
- Correlated faults are more difficult to detect than independent faults.

Software Faults

- Software faults are caused by incorrect specification, design, or coding of a program.
- Although software does not physically "break" after being installed in a computer system, latent faults or bugs in the code can surface during operation - especially under heavy or unusual workloads - and eventually lead to system failures.

Error

- A fault in a system does not necessarily result in an error.
- An error occurs only when a fault is "sensitized", i.e., for a particular system state and input excitation, an incorrect next state and/or output results.
- A error may be latent or detected. An error is latent when it has not been recognized as such; an error is detected by a detection mechanism. An error may disappear before being detected.
- An error may, and in general does, propagate; by propagating, an error creates other (new) error(s).

Error Recovery

- *Error recovery* is the process by which the system attempts to recover from the effects of an error.
  - **Forward error recovery**:
    In this approach, the error is masked without any computations having to be redone.
  - **Backward error recovery:**
    In this approach, the system is rolled back to a state before the error is believed to have occurred and the computation is carried out again.

Failure

- *Failure* denotes an element's inability to perform its designated function because of errors in the element or its environment, which in turn caused by various faults. The ways a system can fail may be classed according to two viewpoints:
- **Mode of failure:**
  - *Fail-controlled:* the mode of failure has been specified and the system complies with this specification.
  - *Fail-uncontrolled*: the mode of failure either does not comply with the specification or has not been specified?

Failure (Contd.)

- **Severities of failure:** The failure severities result from grading the consequences of the failure modes upon the system environment.
  - *benign failures:* where the consequences are of the same order of magnitude as the benefit provided by proper service delivery.
  - *catastrophic failures:* where the consequences are incommensurable with the benefit provided by proper service delivery.

Classes of faults, errors, and failures

- Related faults manifest themselves as similar errors and lead to common-mode failures, whereas independent faults usually cause distinct errors and separate failures.

$$\begin{tikzcd}
\text{Independent faults} \arrow[r] \arrow[dr] & \text{Distinct errors} \arrow[r] \arrow[dr] & \text{Separate failures} \\
\text{Related faults} \arrow[r] & \text{Similar errors} \arrow[r] & \text{Common-mode failures}
\end{tikzcd}$$

Load and Fault Hypothesis

- Any system has a finite processing power. If we intend to guarantee by design that certain performance requirements can be met, then we have to postulate a set of assumptions about the behavior of the environment.
  - **Load hypothesis** defines the peak load that is assumed to be generated by the environment.
  - **Fault hypothesis** defines the types and frequency of faults that a system must be capable of handling.
- The worst scenario that a fault-tolerant system must be capable of handling is at peak load with the maximum number of faults.

Graceful Degradation

- If a specified fault scenario develops, the system must still provide a specified level of service. If more faults are generated than what is specified in the fault hypothesis, then, sometimes, the performance of the system must **degrade gracefully**; i.e., the system must not suddenly collapse as the size of the faults increases, rather it should continue to execute part of the work load.
- The concept of **assumption coverage** defines the probability that the load and fault hypotheses - and all other assumptions made about the behavior of the environment - are in agreement with the reality.

Dependability Measures - Quantitative

- A life of a system is perceived by its users as an alternation between two states of the delivered service: *proper service* and *improper service*.
- A failure is thus a transition from proper to improper service.
  Quantifying the alternation of proper-improper service leads to the two main measures of dependability: *reliability* and *availability*.

Reliability

- *Reliability* is a measure of continuous delivery of proper service - or, equivalently, of the *time* to failure. In other words, it is the probability of surviving (potentially despite failures) *over an interval of time*.
- For instance, the reliability requirement might be stated as a 0.999999 availability for a 10-hour mission. In other words, the probability of failure during the mission may be at most 10^{-6}.
- Hard real-time systems such as flight control and process control demand high reliability, in which a failure could mean loss of life.
-

Availability

- *Availability* is a measure of the delivery of proper service with respect to the alternation of proper and improper service. In other words, it is the probability of being operational *at a given instant of time*.
- A 0.999999 availability means that the system is not operational at most one hour in a million hours.
- It is important to note that a system with high availability may in fact fail. However, failure frequency and recovery time should be small enough to achieve the desired availability.
- Soft real-time systems such as telephone switching and airline reservation require high availability.

Dependability Measures - Qualitative

- **Fail-safe:** Design the system in such a way that one or more safe states can be identified that can be accessed in case of a system failure.
  Example: A railway signaling system in which, on detection of a failure, all trains can be stopped to avoid severe consequences.
- **Fail-operational:** Design the system so that, when it sustains a specified number of faults, it still provides a subset of its specified service. In such systems, safe states cannot be identified. *Example:* A flight control system in which the system must deliver minimal level of service even in the case of failure.

Dependability Measures - Qualitative (Cont'd)

- **No single point of failure**
  - Design the system so that the failure of any single component will not cause the system to fail.
- **Consistency**
  - Design the system so that all information delivered by the system is equivalent to the information that would be delivered by an instance of a non-faulty system.