Characteristics of a RTS

- Large and complex
- Concurrent control of separate system components
- Facilities to interact with special purpose hardware
- Guaranteed response times
- **Extreme reliability**
- Efficient implementation

Reliability and Fault Tolerance

- Goal
  - To understand the factors which affect the reliability of a system and how software design faults can be tolerated
- Topics
  - Reliability, failure and faults
  - Failure modes
  - Fault prevention and fault tolerance
  - N-Version programming
  - Software dynamic redundancy
  - The recovery block approach to software fault tolerance
  - A comparison between N-version programming and recovery blocks
  - Dynamic redundancy and exceptions
  - Safety, reliability and dependability

Scope

- Four sources of faults which can result in system failure
  - Inadequate specification
    - Not covered here
  - **Design errors in software**
    - Covered now
  - Processor failure
    - Not covered here, see B&W book
  - Interference on the communication subsystem
    - Not covered here, see B&W book

Reliability

- The reliability of a system is a measure of the success with which it conforms to some authoritative specification of its behaviour
  - May vary with time
- Very solidly established metrics exist for hardware components
  - Electronic components are observed to fail at a constant rate
- Reliability at time t for those components is modelled by
  - $R(t) = Ge^{-\lambda t}$
    where G is a component-specific constant and λ is the sum of the failure rates of all its constituent components
- The mean time between failures (MTBF) is a commonly used metric (time to failure + time to repair)
  - For a system without redundancy MTBF = 1 / λ

Failure and Faults

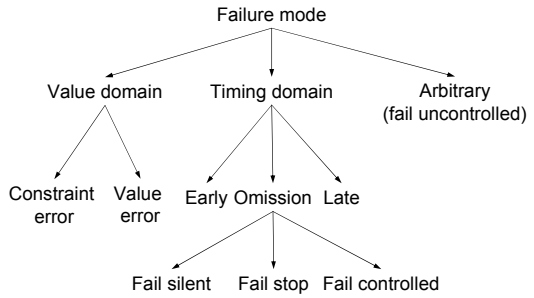
- When the behaviour of a system deviates from what is specified for it, this is called a failure
- Failures result from unexpected problems internal to the system which eventually manifest themselves in the system's external behaviour
- These problems are called errors and their mechanical or algorithmic or conceptual cause are termed faults
- Systems are composed of components which are themselves systems: hierarchically therefore

  Failure] → [Fault → Error → Failure] → [Fault

Fault Types

- Transient faults start at a particular time, remain in the system for some period and then disappear
  - E.g., hardware components which have an adverse reaction to radioactivity
  - Many faults in communication systems are transient
- Permanent faults remain in the system until they are repaired
  - E.g., a broken wire or a software design error
- Intermittent faults are transient faults that occur from time to time
  - E.g., a hardware component that is heat sensitive, it works for a time, stops working, cools down and then starts to work again

Failure Modes



Approaches to Reliability

- Fault prevention attempts to eliminate any possibility of faults creeping into a system before it goes operational
  - Fault avoidance
  - Fault removal
- Fault tolerance enables a system to continue functioning even in the presence of faults
  - Hardware / software fault tolerance
  - Static / dynamic fault tolerance
- Both approaches attempt to produces systems which have well-defined failure modes

Fault Prevention: Fault Avoidance

- Fault avoidance attempts to limit the introduction of faults during system construction by
  - Use of the most reliable components within the given cost and performance constraints
  - Use of thoroughly-refined techniques for interconnection of components and assembly of subsystems
  - Packaging the hardware to screen out expected forms of interference
  - Rigorous, if not formal, specification of requirements
  - Use of proven design methodologies
  - Use of languages with facilities for data abstraction and modularity
  - Use of software engineering environments to help manipulate software components and thereby manage complexity

Fault Prevention: Fault Removal

- In spite of fault avoidance, design errors in both hardware and software components may still exist
- Fault removal uses procedures for finding and removing the causes of errors
  - E.g., design reviews, program verification, code inspections and system testing
- System testing however can never be exhaustive and remove all potential faults
  - A test can only be used to show the presence of faults, not their absence
  - It is sometimes impossible to test under realistic conditions
  - Most tests are done with the system in simulation mode and it is difficult to guarantee that the simulation is accurate
  - Errors that have been introduced at the requirements stage of the system development may not manifest themselves until the system goes operational

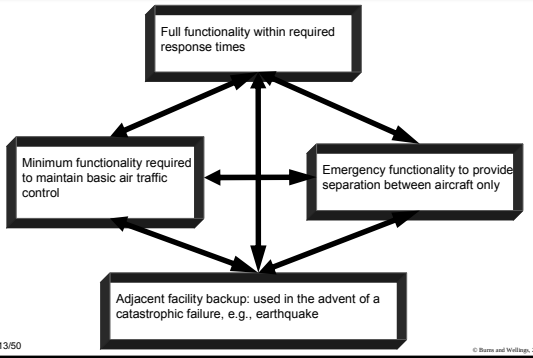
Failure of Fault Prevention Approach

- In spite of all the testing and verification techniques, hardware components will fail
- The fault prevention approach will therefore be unsuccessful when
  - Either the frequency or duration of repair times are unacceptable, or
  - The system is inaccessible for maintenance and repair activities
- In those cases the necessary complement is Fault Tolerance

Levels of Fault Tolerance

- 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 (fail soft)
  - The system continues to operate in the presence of errors, 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
- The level of fault tolerance required will depend on the application
- Most safety-critical systems require full fault tolerance, however in practice many settle for graceful degradation

Graceful Degradation in an ATC System



Redundancy

- All fault tolerance techniques rely on extra elements introduced into the system to detect errors and faults and to recover from them
- Those extra elements are redundant as they are not required in a perfect system
  - Often called protective redundancy
- Minimise redundancy while maximising reliability, subject to the cost and size constraints of the system
  - The added components increase the complexity of the system
  - Which can lead to less reliable systems
- The fault-tolerant components are best separated from the rest of the system

Hardware Fault Tolerance

- Static redundancy (error masking)
  - Redundant components are used inside a system to hide the effects of faults
  - E.g., Triple Modular Redundancy (TMR)
    - 3 identical subcomponents and majority voting circuits
    - The outputs are compared and if one differs from the other two that output is masked out
  - Assumes the fault is not common (such as a design error) but is either transient or due to component deterioration
  - To mask faults from multiple components requires NMR
- Dynamic redundancy (error detection)
  - Error detection facility supplied inside a component indicates that the output is in error
  - Recovery must be provided by another component
  - E.g., communication checksums and memory parity bits

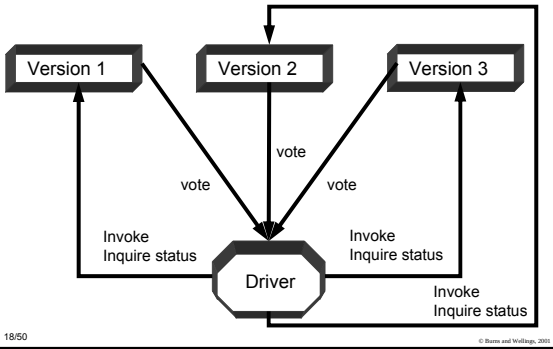
Software Fault Tolerance

- Used for detecting design errors
- Static fault tolerance
  - N-Version programming
    - Software equivalent to NMR
- Dynamic fault tolerance
  - Detection and Recovery
  - Recovery blocks: backward error recovery
  - Exceptions: forward error recovery

N-Version Programming – 1

- Design diversity
  - The independent generation of N (N > 2) functionally equivalent programs from the same initial specification
  - No interactions between development groups
  - The programs execute concurrently with the same inputs and their results are compared by a driver process
  - The results (assimilated to VOTES) should be identical
  - If they are not the consensus result, assuming there is one, is taken to be correct

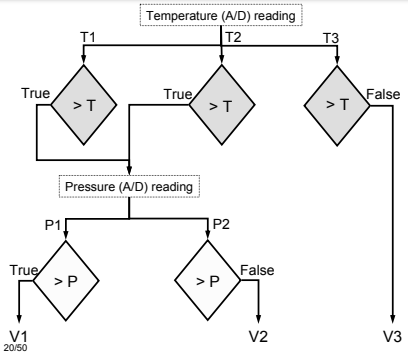
N-Version Programming – 2



Vote Comparison

- To what extent can votes be compared?
  - Far from obvious
- Text or integer arithmetic will produce identical results
  - Voting can on equality
- Real numbers will produce different values
  - Need inexact voting techniques
- User defined types outside of numeric will need their own equality
  - Limited types in Ada

Consistent Comparison Problem



Each version will produce a different result, but correct within finite-precision arithmetic

Even using inexact voting, the problem occurs when the values are close to the decision threshold

N-version Programming – 3

- Initial specification
  - The majority of software faults stem from inadequate specification? A specification error will manifest itself in all N versions of the implementation
- Independence of effort
  - Experiments produce conflicting results. Where part of a specification is complex, this leads to a lack of understanding of the requirements. If these requirements also refer to rarely occurring input data, common design errors may not be caught during system testing
- Adequate budget
  - The predominant cost is software. A 3-version system will triple the budget requirement and cause problems of maintenance. Would a more reliable system be produced if the resources potentially available for constructing an N-versions were instead used to produce a single version?

Software Dynamic Redundancy

- Error detection
  - No fault tolerance scheme can be utilised until the associated error is detected
- Damage confinement and assessment
  - To what extent has the system been corrupted?
  - The delay between a fault occurring and the detection of the error means erroneous information could have spread throughout the system
- Error recovery
  - Techniques should aim to transform the corrupted system into a state from which it can continue its normal operation (perhaps with degraded functionality)
  - Probably the most important phase of any fault tolerance technique
- Fault treatment and continued service
  - An error is a symptom of a fault
  - Although damage repaired, the fault may still exist

Error Detection

- Environmental detection
  - Hardware
    - E.g., illegal instruction
  - OS / run-time support
    - E.g., Null pointer
- Application detection
  - Replication checks
  - Timing checks
  - Reversal checks
  - Coding checks
  - Reasonableness checks
  - Structural checks
  - Dynamic reasonableness check

Damage Confinement and Assessment

- Damage assessment is closely related to damage confinement techniques used
- Damage confinement is concerned with structuring the system so as to minimise the damage caused by a faulty component (also known as firewalling)
- Modular decomposition provides static damage confinement
  - Allows data to flow through well-define pathways
- Atomic actions provides dynamic damage confinement
  - They are used to move the system from one consistent state to another

Forward Error Recovery

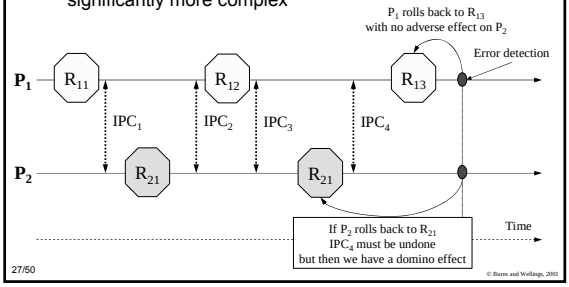
- Forward error recovery continues on from an erroneous state by making selective corrections to the system state
- This includes making safe the controlled environment which may be hazardous or damaged because of the failure
- It is system specific and depends on accurate predictions of the location and cause of errors (i.e, damage assessment)
- Examples
  - Redundant pointers in data structures
  - Use of self-correcting codes such as Hamming Codes

Backward Error Recovery

- Backward error recovery relies on restoring the system to a previous safe state and executing an alternative section of the program
- This has the same functionality but uses a different algorithm (c.f. N-Version Programming) and therefore no fault
- The point to which a process is restored is called a recovery point and the act of establishing it is termed check-pointing (saving appropriate system state)
  - The erroneous state is cleared and it does not rely on finding the location or cause of the fault
  - Can therefore be used to recover from unanticipated faults including design errors
  - But it cannot undo errors in the environment!

The Domino Effect

- With concurrent processes that interact with one another, Backward Error Recovery becomes significantly more complex



Fault Treatment and Continued Service

- Error recovery returned the system to an error-free state; however, the error may recur; the final phase of fault tolerance is to eradicate the fault from the system
- The automatic treatment of faults is difficult and system specific
- Some systems assume all faults are transient; others that error recovery techniques can cope with recurring faults
- Fault treatment can be divided into 2 stages
  - Fault location
  - System repair
- Error detection techniques can help to trace the fault to a component
  - For hardware the component can be replaced
  - A software fault can be removed in a new version of the code
  - In non-stop applications it will be necessary to modify the program while it is executing!

Recovery Block – 1

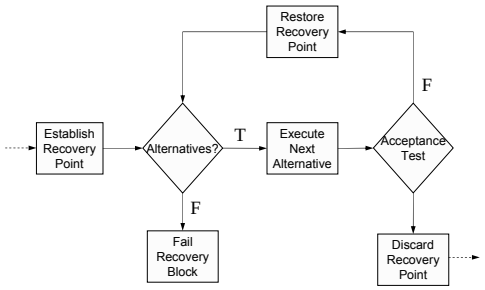
- Language support for backward error recovery
- At the entrance to a block is an automatic recovery point and at the exit an acceptance test
- The acceptance test is used to test that the system is in an acceptable state after the block's execution
  - Primary module
- If the acceptance test fails, the program is restored to the recovery point at the beginning of the block and an alternative module is executed
- If the alternative module also fails the acceptance test, the program is restored to the recovery point and yet another module is executed, and so on
- If all modules fail then the block fails and recovery must take place at a higher level

Recovery Block – 2

```
ensure <acceptance test>
by
   <primary module>
else by
   <alternative module>
else by
   <alternative module>
...
else by
   <alternative module>
else error
```

- Recovery blocks can be nested
- If all alternatives in a nested recovery block fail the acceptance test, the outer level recovery point will be restored and an alternative module to that block executed

Recovery Block – 3



The Acceptance Test

- The acceptance test provides the error detection mechanism which enables the redundancy in the system to be exploited
- The design of the acceptance test is crucial to the efficacy of the Recovery Block scheme
- There is a trade-off between providing comprehensive acceptance tests and keeping overhead to a minimum, so that fault-free execution is not affected
- Note that the term used is acceptance, not correctness
  - This allows a component to provide a degraded service
- All the previously discussed error detection techniques discussed can be used to form the acceptance tests
- However, care must be taken as a faulty acceptance test may lead to residual errors going undetected

N-Version Programming vs. Recovery Blocks

- Type of redundancy
  - NVP is static, RB is dynamic
- Design overheads
  - Both require alternative algorithms
    - NVP requires driver, RB requires acceptance test
- Run-time overheads
  - NVP requires ×N resources
  - RB requires establishing recovery points
- Diversity of design
  - Both susceptible to errors in requirements
- Error detection
  - Vote comparison (NVP) vs. acceptance test (RB)
- Atomicity
  - NVP vote before it outputs to the environment
  - RB must be structured to only output after passing an acceptance test

Dynamic Redundancy and Exceptions

- An exception can be defined as the occurrence of an error
- Bringing an exception to the attention of the invoker of the operation which caused the exception, is called raising (or signalling or throwing) the exception
- The invoker's response is called handling (or catching) the exception
- Exception handling is a forward error recovery mechanism as there is no roll back to a previous state
  - Control is passed to the handler so that recovery procedures can be initiated
- However, the exception handling facility can also be used to provide backward error recovery

Exceptions – 1

- Exception handling can be used to
  - Cope with abnormal conditions arising in the environment
    - The original motivation
  - Enable program design faults to be tolerated
    - Not the original intent with exceptions!
  - Provide a general-purpose error detection and recovery facility

Exceptions – 2

- Requirements for an exception handling facility
  1) The facility must be simple to understand and use
  2) The code for exception handling should not obscure understanding of the program's normal error-free operation
  3) The mechanism should be designed so that run-time overheads are incurred only when handling an exception
  4) The mechanism should allow the uniform treatment of exceptions detected both by the environment and by the program
  5) the exception mechanism should allow recovery actions to be programmed

Exceptions – 3

- Two sources of detection
  - Environmental detection
  - Application error detection
- A synchronous exception is raised as an immediate result of a process attempting an inappropriate operation
- An asynchronous exception is raised some time after the operation causing the error
  - It may be raised in the process which executed the operation or in another process
- Asynchronous exceptions are often called asynchronous notifications or signals

Exceptions – 4

- There are two models for the declaration of synchronous exceptions
  - A constant name which needs to be explicitly declared (in Ada)
  - An object of a particular type which may or may not need to be explicitly declared (in Java)
- In Ada the exceptions that can be raised by the run-time support are declared in package Standard

```
package Standard is
 ...
 Constraint_Error : exception;
 Program_Error : exception;
 Storage_Error : exception;
 Tasking_Error : exception;
 ...
end Standard;
```

- This package is visible to all Ada programs

Exceptions – 5

- Within a program, there may be several handlers for a particular exception
- Associated with each handler is a domain which specifies the region of computation during which, if an exception occurs, the handler will be activated
- The accuracy with which a domain can be specified will determine how precisely the source of the exception can be located

Exceptions – 6

- If there is no handler associated with the block or procedure
  - Regard it as a programmer error to be reported at compile time
  - But an exception raised in a procedure and not handled in it, can only be handled within the context from which the procedure was called
    - E.g., an exception raised in a procedure as a result of a failed assertion involving the parameters
- CHILL requires that a procedure specifies which exceptions it may raise (that is, not handle locally)
  - The compiler can then check the calling context for an appropriate handler
- Java allows a function to define which exceptions it can raise
  - However, unlike CHILL, it does not require a handler to be available in the calling context

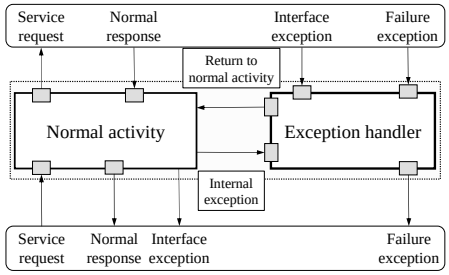
Exceptions – 7

- Otherwise look for handlers up the chain of invokers
  - This is called propagating the exception
  - The Ada and Java approach
- A problem occurs where exceptions have scope
  - An exception may be propagated outside its scope, which makes it impossible for a handler to be found
- Most languages provide a catch-all exception handler
- An unhandled exception causes a sequential program to be aborted
- If the program contains more than one process and a particular process does not handle an exception it has raised, then usually that process is aborted
  - However, it is not clear whether the exception should be propagated to the parent process

Exceptions – 8

- Should the invoker of the exception continue its execution after the exception has been handled?
- If the invoker can continue, then it may be possible for the handler to cure the problem that caused the exception to be raised and for the invoker to resume as if nothing has happened
  - This is referred to as the resumption or notify model
- The model where control is not returned to the invoker instead is called termination or escape
- Clearly it is possible to have a model in which the handler can decide whether to resume the operation which caused the exception, or to terminate the operation
  - This is called the hybrid model

Ideal Fault-Tolerant Component



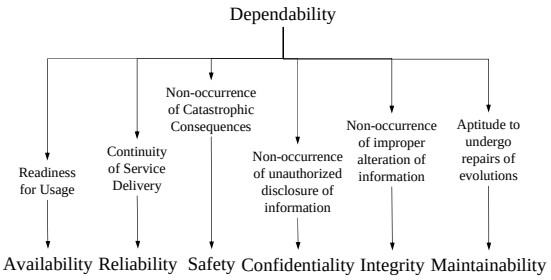
Safety – 1

- Safety
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss of) equipment (or property), or environmental harm
  - By this definition, most systems which have an element of risk associated with their use are unsafe
- A mishap is an unplanned event or series of events that can result in death, injury, etc.
- Safety is the probability that conditions that can lead to mishaps do not occur whether or not the intended function is performed
- How does that relate to reliability?

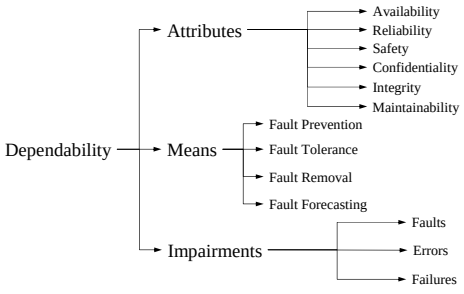
Safety – 2

- Measures which increase the likelihood of a weapon firing when required may well increase the possibility of its accidental detonation
  - Aiming at better reliability may decrease safety
- In many ways, the only safe airplane is one that never takes off, however, it is not very reliable
  - Aiming at greater safety may decrease reliability
- As with reliability, to ensure the safety requirements of an embedded system, system safety analysis must be performed throughout all stages of its life cycle development

Dependability Aspects



Dependability Terminology



Summary – 1

- Reliability
  - A measure of the success with which the system conforms to some authoritative specification of its behaviour
- When the behaviour of a system deviates from that which is specified for it, this is called a failure
- Failures result from faults
- Faults can be accidentally or intentionally introduced into a system
- They can be transient, permanent or intermittent
- Fault prevention consists of fault avoidance and fault removal
- Fault tolerance involves the introduction of redundant components into a system so that faults can be detected and tolerated

Summary – 2

- N-version programming
  - The independent generation of N (where N >= 2) functionally equivalent programs from the same initial specification
- Based on the assumptions that a program can be completely, consistently and unambiguously specified, and that programs which have been developed independently will fail independently
- Dynamic redundancy
  - Error detection, damage confinement and assessment, error recovery, and fault treatment and continued service
- Atomic actions to aid damage confinement
  - Not discussed here

Summary – 3

- With backward error recovery, it is necessary for communicating processes to reach consistent recovery points to avoid the domino effect
- For sequential systems, the recovery block is an appropriate language concept for backward error recovery
- Although forward error recovery is system specific, exception handling has been identified as an appropriate framework for its implementation
- The concept of an ideal fault-tolerant component was introduced which uses exceptions
- The notions of software safety and dependability have been introduced