Software Reliability and Security

Module 7

Winter 2017

Presentation/Lecture Schedule and Report Due Dates

- Presentation 1
  - Related background paper
  - Jan 27, Feb 1, 3
- Presentation 2
  - Project proposal
  - March 1, 3, 8
- Presentation 3
  - Final project report
  - March 24, 29, 31

- Lectures

  Jan 13, 18, 20, 25, 27

  Feb 1, 3, 8, 10, 15, 17

  March 1, 3, 8, 10, 15, 17, 22, 24, 29, 31

- Project Proposal Due
  Tuesday, February 28

- Final Project Report Due
  Monday, April 10

- Final Exam
  Wednesday, April 12, 10:00am

Outline

- Dependability – A Generic Concept
  - Attributes
  - Impairments
  - Means
- The Impairments to Dependability
  - Faults, errors, and failures
  - Classifications of faults, errors, and failures
- Means for Dependability – Fault Tolerance
  - Phases of fault tolerance
  - Approaches for software fault tolerance

Fault Tolerance

- Fault tolerance - the behavior meets the specifications despite the failure of one or more of its components
  - Can be made fault tolerant against only the failure of its components – not against the system failure
- Fault tolerance is achieved by redundancy
  - Redundancy - the parts that are not needed for the correct (normal or without fault tolerance support) operation of the system
  - Fundamental assumption – redundant and regular components usually do not fail at the same time

Types of Redundancy

- Hardware redundancy
  - Mainly to tolerate hardware faults – out of the scope of this course
  - Example s– extra processor, memory, communication links or HW needed to run redundant software
- Software redundancy
  - Mainly to tolerate software faults – discussed in this course
  - Redundant software components on the same hardware or software needed to control any redundant HW
- Time redundancy
  - Extra time allowed for the tasks to support fault tolerance
  - Execute some instruction(s) multiple (redundant) times

Fault Tolerance Phases

- Detection phases
  1. Error Detection – a failure of the component (discussed further)
  2. Damage Confinement and Assessment
- Recovery phases
  3. Error Recovery – bringing the system to an error-free state (discussed further)
  4. Fault Treatment and Continued Service

Phases of Fault Tolerance

- Damage Confinement and Assessment
  - Any damage due to the failure has to be identified and removed
  - Dynamically - record and examine the information flow
  - Statically – design the system with "fire walls" which blocks information flow

- Fault Treatment and Continued Service
  - The fault is identified (fault localization)
  - Avoid using the faulty components or use them in a way so that the failures are not repeated (system repair)

Error Detection

- Ideal error detection attributes
  - Solely from the specification (black box) - knowledge of internal design can cause the same error in the detection as it is present in the system
  - Complete and correct – all errors and no spurious errors detected
  - Independent from the system – both the detector and the system should not fail at the same time

Error Detection – Some General Types of Checks

- Replication checks
  - Replicate some components of the system
  - The results of different components are compared or voted
  - Example: triple modular redundancy (TMR) for hardware fault tolerance
- Timing checks
  - Any component timing constraints can be checked by setting timer with the specified (expected) value – watchdog timers
- Structural and coding checks
  - Semantic – the meaning or value is consistent with the rest of the system?
  - Structural – the structure of the data is as it should be - usually employed for hardware

Error Detection – Some General Types of Checks – contd.

- Reasonableness check
  - States of objects in the system are "reasonable"? - range check, assertion check
- Diagnostics checks
  - The check is performed by the system on its components (in other checks, the detection task was part of a detection system)
  - Typically some special input values are fed to a system and compared with the known outputs – usually used in systems at power-up time

Error Recovery

- Error Recovery
  - Backward Recovery
  - Forward Recovery
- Backward Recovery
  - The sate of the system is periodically check-pointed
  - When an error or failure is detected, the system is rolled back to its check-pointed state (assuming that state is error-free)
  - Overhead involved: check-pointing and rollback

Error Recovery – contd.

- Forward Recovery
  - No previous state is available, and the system does not roll back
  - The system is moved forward to an error-free state by taking appropriate corrective actions
  - Less overhead - requires an accurate set of actions to remove the errors occurred
  - Less common than backward recovery

Some Software Fault Tolerance Approaches

- Exception Handling
- N-Version Programming
- Recovery Block

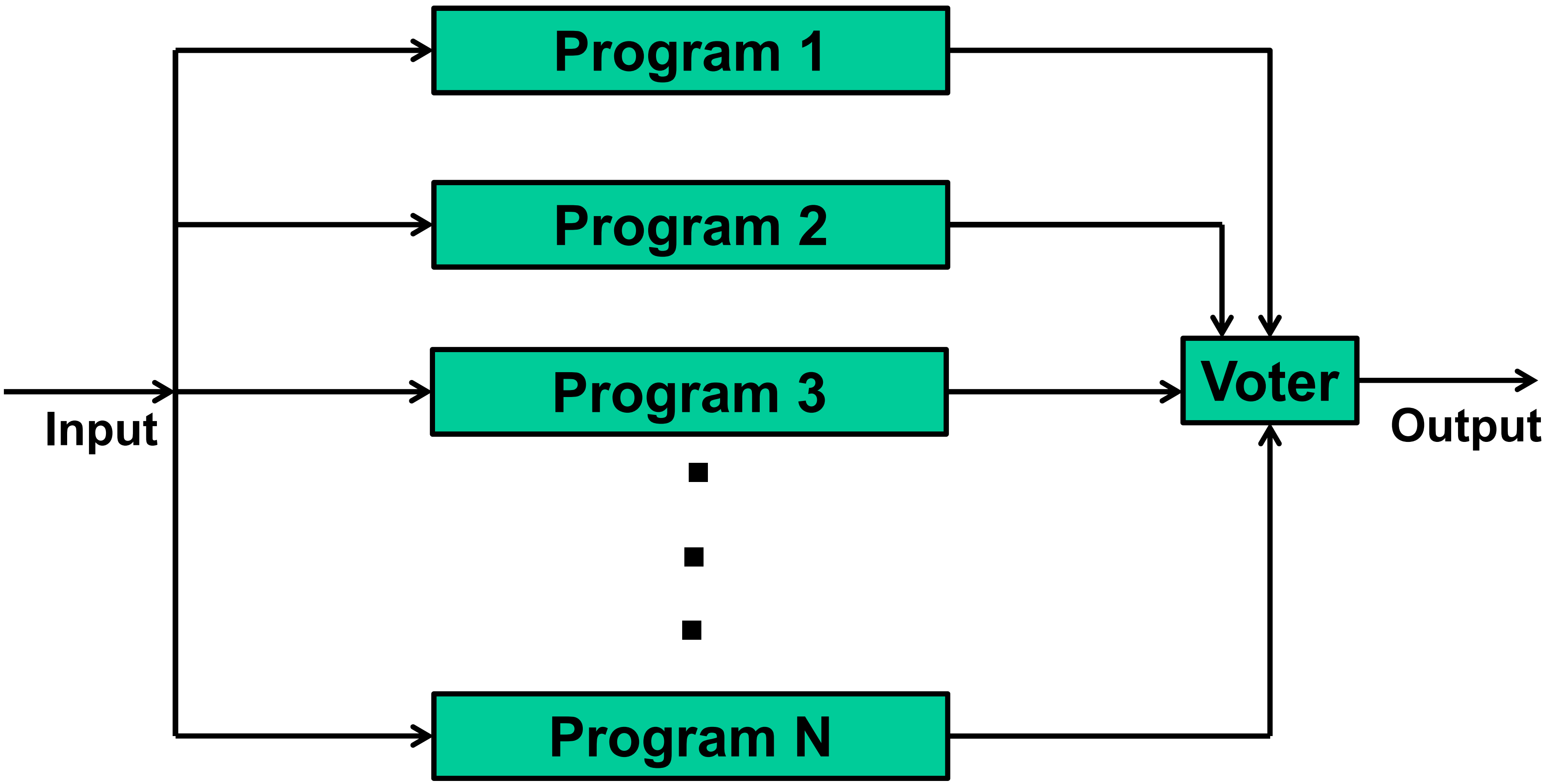
Programmed Exception Handling

- Exception - If a component is called in an abnormal state
  - Anticipated exceptions
  - Unexpected exceptions
- Provides language primitives for signaling and handling exceptions
  - A typical situation: if a called module cannot perform a task correctly, it signals to the caller module to handle the task at problem
- The handler may use backward or forward recovery
- Java exceptions???

N Version Programming Approach

- Tolerates software faults employing multiple versions of the same software
- A number of versions (>2) of the same software are developed
- All versions run simultaneously from same states and inputs
- The outputs from different versions go to a voting algorithm that selects the correct output

N Version Programming – contd.



N-Version Programming – contd.

- Primary goal: handle design faults if the separate versions are designed /implemented to meet the same specifications (same required functionalities)
- Independent (isolation) development of each version – avoid the introduction of similar or identical faults
- Value of *N*
  - Depends on voting algorithm and failure types
  - For a *t* fault tolerant system the number of versions required
    - Fail-stop failures: *t+1*
    - Byzantine failures: *2t+1* (assuming majority *t+1* implementations are correct)

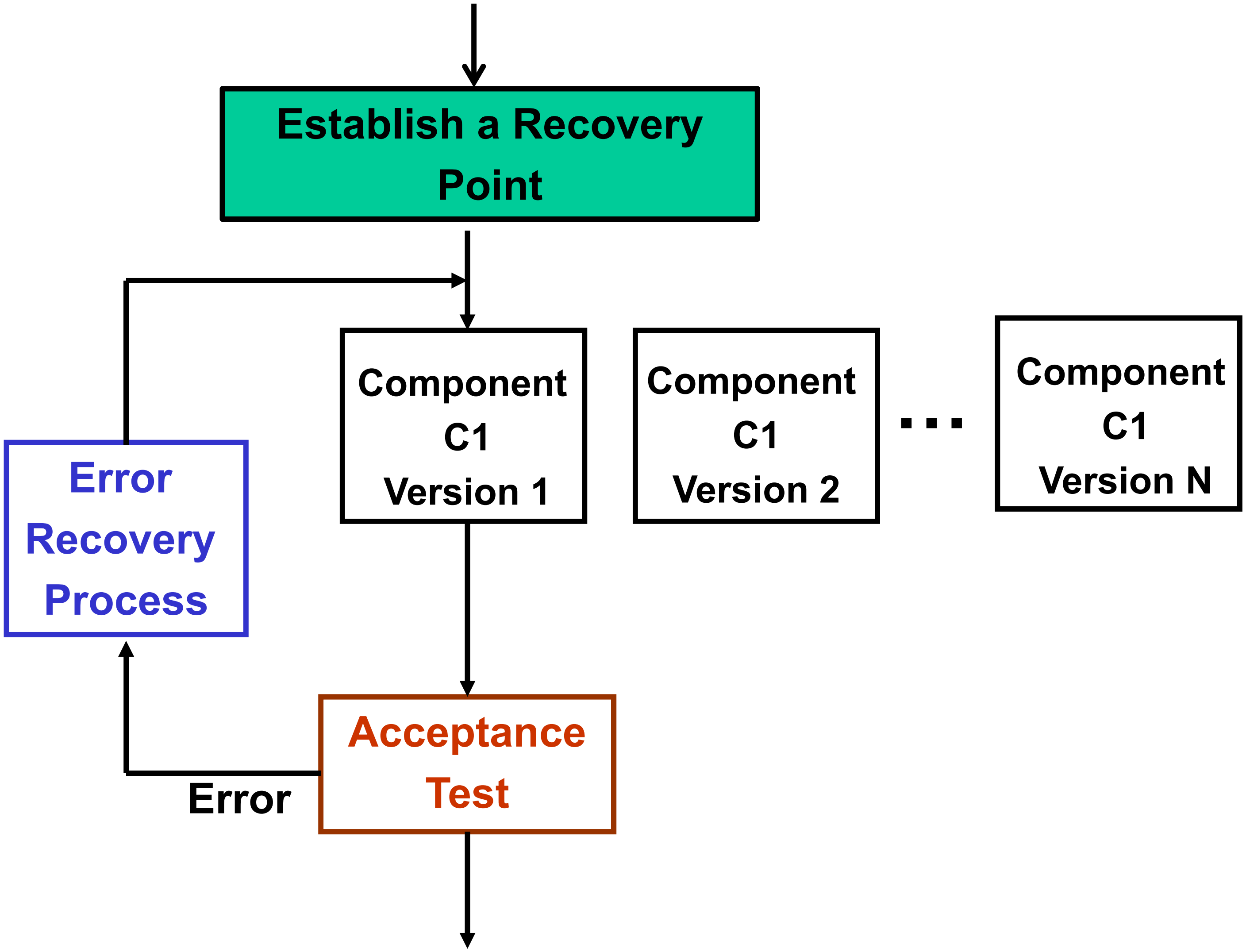
Voting Algorithms – Range of Correct Outputs

- Majority Voting: m-out-of-N Voting
  - Assumption: Majority of the N results are the same and correct
  - Required: m>= ceiling of (N+1)/2
- Consensus Voting
  - The majority of identical results are selected as correct output
  - There may be more than one identical majority – choose one of them
- Two-out-of-N Voting
  - A special case of consensus voting
  - Assumption: Two versions don't generate wrong & identical results
  - Sometimes faster than majority and consensus voting – can make a decision as soon as it gets two identical results

Recovery Block Approach

- Tolerate software faults by employing multiple versions of some components – redundant components
- Each implemented module is designed and implemented independently to meet the same I/O requirements
- Check pointing is essential
- The correctness of each component output is tested through acceptance testing
- Requires effective error recovery process

Recovery Block


Establish a Recovery Point

Component C1 Version 1

Component C1 Version 2

. . .

Component C1 Version N

Error Recovery Process

Acceptance Test

Error

Recovery Block Scheme

- A recovery point is check pointed before executing a component (primary component)
- The primary component is run and its output is checked using acceptance testing
- If an error is detected, the system is brought back to a state (backward recovery)
- The software is run using another version of the same component and the acceptance test is repeated
- If all components fail the acceptance testing, the system reaches an erroneous state

Recovery Block – An Example by Program

```
ensure $A[i+1] >= A[i]$ for $i=1, 2, \ldots, n-1$
by
    sort $A$ using quick sort
else by
    sort $A$ using shell sort
else by
    sort $A$ using bubble sort
else error
```

Nested Recovery Block - Example

```
ensure   <acceptance test for outer recovery block >
by       <primary block >
          .
          .
          .
          ensure  <acceptance test for inner recovery block >
          by <primary module>
          else by <alternate module>
          else error
          .
          .
          .
else by   <another module>
          .
          .
          .
else error
```

Recovery Block VS N-Version Programming

- Recovery Block
  - Fault tolerance is at key component levels where each component version is run sequentially
  - Provides the option of replicating only key components – replicating the whole system is expensive
  - Variation in execution time – may be difficult for real-time applications but suitable for gracefully degrading systems (correct output with different response time)
  - Acceptance test
    - limited coverage than general error detection test
    - useful when multiple correct outputs are possible – e.g., finding root of an equation

Recovery Block VS N-Version Programming – contd.

- N Version Programming
  - Fault tolerance is at program level where each version is run in parallel
  - Voting may have more detection coverage than acceptance testing
  - Voting check in SW is more difficult than in HW– has to deal with floating point number and voting algorithm tolerance is an issue
  - Voting algorithm does not work when there are multiple acceptable results – roots of equation, non-deterministically specified system
  - Independence assumption of design diversity is difficult to achieve – different versions tend to have similar faults
  - Design diversity increases the software development cost significantly
  - Very hard to handle SW engineering issues (project management, development , maintenance, etc.)

Summary

- Concept of Dependability
  - Attributes, impairments, and means
- The Impairments to Dependability
  - Faults, errors, and failures
  - Classifications of faults, errors, and failures
- Phases of Fault Tolerance
  - Error Detection
  - Damage Confinement and Assessment
  - Error Recovery
  - Fault Treatment and Continued Service
- Software Fault Tolerance Approaches
  - Exception Handling
  - Recovery Block
  - N-Version Programming

Lecture Sources

- J.C. Laprie, Dependabilty: Basic Concepts and Terminology in English, French, German, Italian and Japanese, Springer-Verlag, NY, 1991.
- J.C. Laprie, Dependable Computing and Fault Tolerance: Concepts and Terminology, FTCS-15, IEEE 1985.
- Pankaj Jalote, Fault Tolerance in Distributed Systems, Prentice-Hall, New Jersy, 1998.
- P.A. Lee & T. Anderson, Fault Tolerance Principles and Practice, Springer-Verlag, 1990.