
SYSC 3303 Real-Time Concurrent Systems

Verification and Validation

- Copyright © 2001-2003 D.L. Bailey, 2003-2012 L.S. Marshall, Systems and Computer Engineering, Carleton University
- revised August 9th, 2012

Resources

- Most books on software V&V (there aren't many) have little to say about real-time, concurrent, and distributed systems
- Most books on real-time concurrent systems (there aren't many) have little to say about testing
- Papers in refereed journals and conference proceedings are probably your best source of information about techniques and tools

Sequential Program V&V Techniques

- Each thread/task/lightweight process (lwp) is not concurrent
 - concurrency is between the threads/tasks/lwps
- This means that you can use classic V&V techniques for sequential programs that you learned in previous courses to test individual threads
 - assertions, invariants, boundary conditions, etc.
 - additional testing techniques will be taught in 4th-year courses

Sequential Program V&V Techniques

- If thread responsibilities are allocated to modules/units/classes (depending on the programming technology), these can be tested in a standalone manner
 - unit testing, integration testing
 - drivers, stubs

Concurrent System V&V Isn't Easy

- Even if you thoroughly test the individual modules and classes that comprise a concurrent system, don't be surprised if the system doesn't work when you first link the collection of components together
- The reason for this goes back to the fundamental nature of real-time systems: they must respond to *unpredictable* stimuli
- The s/w developer cannot predict when events will arrive from the environment (or when expected events fail to arrive) relative to the state of the program's execution

Concurrent System V&V Isn't Easy

- Low-level hardware interrupts occur, causing interrupt service routines to run, which in turn make I/O blocked threads ready to run; and threads are scheduled according to their priorities
- *A real-time program never runs exactly the same way twice!*
- Bugs mysteriously disappear and reappear
- What can we do about this?

Let the Maps Guide the Implementation

- Modern software development processes recognize the importance of following an incremental, iterative approach to constructing software systems
 - you learned about this in earlier courses
- This holds for real-time, concurrent systems
- We recommend using a staged approach based on UCMs, in which we implement complete paths through a set of *incomplete* components

Let the Maps Guide the Implementation

- Confidence testing
 - code and test one or more prototypes to exercise the major paths that span the entire system
 - refine the prototypes to implement the minor paths
- The idea is that we concentrate first on getting things working along critical paths, and not on making components complete
- This is what you are doing in the project iterations

Ensuring Repeatable Behaviour

- To make the behaviour of the prototypes predictable and repeatable, sources of external stimuli are not coupled into the prototypes until late in the process
- For example, initial testing of the TFTP system involves running the client and server on the same host
- We need some way to simulate network errors (dropped packets, delayed packets, duplicated packets) in a predictable way

Deadlocks and Races

- There are techniques for determining if a system is deadlock-free (these are usually presented in introductory operating systems courses)
- Proving that a system is free of critical races is difficult (impossible?)
- UCMs can help us reason about system-wide behaviour (what-if scenarios)
- It's easier to deal with deadlock and races during high-level design, instead of during low-level code debugging

Event Monitoring

- What if, despite your best efforts, bugs remain?
- System failures in the field may not be reproducible in the lab
- To address this, some systems have *event monitoring* in the application code and/or the run-time system (thread context switcher, IPC mechanisms)
- Key events are logged (to memory or disk) or sent via a link to another computer
- If a bug occurs, the event log provides history that may help track down the problem
- But, watch out for “heisenbugs” due to Heisenberg’s observer effect (the act of observing a system inevitably alters its state)!

Timing Constraints

- A correctly operating hard real-time system must always meet its timing constraints
 - we must be able to validate this
- In contrast, we do not have to validate that soft real-time systems meet timing constraints
 - often sufficient to demonstrate that the system meets some statistical constraint (e.g., average # of missed deadlines/minute is 2 or less)
- Hard real-time == guaranteed service
- Soft real-time == best-effort service

Feasibility Analysis

- Techniques for proving that every thread in a real-time system meets its timing constraints is beyond the scope of this course (this material is typically taught in graduate courses)
- If you're interested, *Real-Time Systems*, Jane Liu, Prentice-Hall, Inc., 2000, provides thorough coverage of the mathematical techniques