Teaching Testing of Interrupt and Exception Handling Code

W. Morven Gentleman

Dalhousie University

Morven.Gentleman@dal.ca

Context of Topic

- Conventional view of programming:
  - Deterministic flow of control, expressed by the programmer
  - Made up of:
    - Sequential execution
    - Conditional execution
    - Loops
    - Procedure invocation

Interrupt and Exception Handler

- Control flow triggered by events
- Interrupts
  - – Event possibly unrelated to thread of control
- Exceptions
  - – Event may be synchronized to thread of control
- This commonly leads to multi-thread concurrency or to state machines or to …

Niche, but significant

- OS and device driver products
- Embedded systems, computers that control and monitor physical systems, are all about interaction with the external world: even cell phones and microwave ovens
  - External world may not follow the normal case
  - Typically 75% or more of code in an embedded system is exception handling

Testing Challenges

- Reliability requirements are often high
- This kind of programming is unfamiliar to many: they make unusual mistakes
- The actions and output are data and time dependent, so often appear to be irreproducible
- Often the consequences are not directly observable

Teaching Testing Challenges

- Need to understand what kinds of defects to look for
- Need to provoke problematic situations
- Need to become facile with appropriate tools but more, to understand what to use them for
- Standard software system abstractions hide details to which access is needed
- Many students are novices to this level of computing

Typical Interrupt Defects

- Overlooked: no code for this situation
- Race conditions
- Activation too late or too early
- Missed interrupts
- Interrupt not cleared
- Inappropriate relative priority or masking
- Improper software runtime interface

Typical Exception Defects

- Overlooked: no sensor to raise the exception
- Overlooked: no code to handle this situation
- Handler for wrong exception
- Wrong handler for particular exception instance
- Invalid attempt to resume an operation
- Invalid attempt to retry an operation
- Invalid attempt to terminate an operation
- Inappropriate cleanup after terminating operation

Why Use Exceptions Anyway?

- *Exception handling:issues and a proposed notation*, John B Goodenough, CACM V18N12 Dec 1975, pp. 683-696
  - To deal with impending or actual failure
    - Range failure or domain failure
  - To indicate the significance of a valid result
  - To permit an invoker to monitor an operation
- In short, exception are not needed just for bugs!

Exception Complications

- Circumlocution to get around language
  - Many languages only support operation abort
  - No exception parameter in current languages
  - Poppng many stack levels incurs overhead
  - No concept of deferred processing
  - No provision for multi-thread

Black Box vs White Box vs ?

- What situations warrant investigation?
- Hard to trigger interrupt or exception situation
- Hard to show a critical race does NOT exist
- Dies conventional white box approach help?
- Exploratory white box?

Testing Tools

- Simulators and Emulators
  - Virtual Machines
  - ICE
    - Sequential triggering
  - External hardware
    - Computer(s) simulating external world
    - synchronized with thread (US patent 6167479)
- Debugger (especially scriptable)

More Testing Tools

- Circular history buffers
- Bit vectors to record where control has been
- Audit routines (concurrently) traversing data structures performing consistency and validity checks
- JVM or CLI bytecode pattern match and transform
- Commercial GUI test tools don’t seem to help

Terminology

- Disambiguating interrupts
  - A physical interrupt may not correspond to a single logical interrupt
- Spurious interrupt
  - An interrupt from an unrecognized source
- Unanticipated interrupt
  - An interrupt which should not happen in this context
- Deferred interrupt handling
  - Interrupt not processed immediately because of masking or preemptive priority

Terminology (cont.)

- First Level and Second Level Handlers
  - Also called upper and lower halves
  - Used for some hardware and some OS designs
  - Handler has two parts, only first part triggered by hardware interrupt. Second part run by high priority thread below hardware priority level