Testing Error Handling Code in Device Drivers Using Characteristic Fault Injection

Jia-Ju Bai, Yu-Ping Wang, Jie Yin, Shi-Min Hu
Department of Computer Science and Technology
Tsinghua University
Beijing, China
DRIVER INTRODUCTION

Role

- Manage hardware devices
- Support high-level programs
- Run in kernel mode
DRIVER ERROR HANDLING

- Occasional errors
  - Kernel exceptions (-ENOMEM, -EFAULT, ....)
  - Hardware malfunctions (-EIO, -EBUSY, ....)
  - ...

- Challenges for error handling
  - Complex program logic and context
  - Many different kinds of errors
  - Infrequent to trigger
  - ...

Error handling code in drivers is necessary but hard to correctly implement
MOTIVATION

- Error handling code is incorrect in some drivers

```
Path: linux-3.1.1/drivers/net/bnx2.c

7869. static int __devinit bnx2_init_board(....)
7870. {
......
7885.   bp->temp_stats_blk = kzalloc(...);
......
7906.   rc = pci_request_regions(pdev, ...);
......
7937.   bp->regview = ioremap_nocache(...);
7938.   if (!bp->regview) {
7939.     dev_err("Cannot map register space, aborting\n");
7940.     rx = -ENOMEM;
7940.     goto err_out_release;
7941.   }
......
8247.   err_out_release:
8248.   pci_release_regions(pdev);
8249.   err_out_disable:
8250.   pci_disable_device(pdev);
8251.   pci_set_drvdata(pdev, NULL);
8252.   err_out:
8253.   return rc;
8254. }
```
MOTIVATION

Patch study
- Source: Patchwork (http://patchwork.ozlabs.org/)
- July 2015

<table>
<thead>
<tr>
<th>Driver Class</th>
<th>Accepted Patches</th>
<th>Error Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C</td>
<td>29</td>
<td>13(44.83%)</td>
</tr>
<tr>
<td>PCI</td>
<td>38</td>
<td>13(34.21%)</td>
</tr>
<tr>
<td>PowePC</td>
<td>42</td>
<td>11(26.19%)</td>
</tr>
<tr>
<td>RTC</td>
<td>24</td>
<td>8(33.33%)</td>
</tr>
<tr>
<td>Network</td>
<td>598</td>
<td>253(42.31%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>731</strong></td>
<td><strong>298(40.77%)</strong></td>
</tr>
</tbody>
</table>

Findings
- 40% of accepted patches are related to error handling code
- Many error handling patches are used to fix common bugs

Error handling code in current drivers is not reliable enough
GOAL

- Testing error handling code in device drivers
  - Bug-detection capability
  - Error-handling-code coverage
  - Automation and efficiency
  - Scalability and generality
BASIC TECHNIQUE

- Software fault injection (SFI)
  - Good coverage for error handling code
  - Exact runtime information for bug detection
  - Support most drivers

Typical SFI System

- Data Analyzer
- Controller
- Fault Library
- Fault Injector
- Workload Library
- Workload Generator
- Runtime Monitor
- Target Driver
PREVIOUS SFI APPROACHES

- Some famous approaches
  - Linux Fault Injection Capabilities Infrastructure
  - ADFI (ISSTA ’15), KEDR (ICST ’11), LFI (DSN ’09), ......

- Limitations
  - Low fault representativeness
  - Numerous redundant test cases
  - Several kinds of faults
  - Much manual effort

Our solution is to introduce driver characteristics into SFI
CHARACTERISTIC 1

- Function return value trigger
  - The error handling code is often triggered by a bad function return value

- Driver study
  - 75% of “goto” statements are in if branches of bad function return values

<table>
<thead>
<tr>
<th>Driver Class</th>
<th>Number</th>
<th>“Goto” Statement</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless</td>
<td>116</td>
<td>5109</td>
<td>3757(73.54%)</td>
</tr>
<tr>
<td>Ethernet</td>
<td>219</td>
<td>6749</td>
<td>5192(76.93%)</td>
</tr>
<tr>
<td>Block</td>
<td>56</td>
<td>1322</td>
<td>1005(76.02%)</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>21</td>
<td>121</td>
<td>89(73.56%)</td>
</tr>
<tr>
<td>Clock</td>
<td>117</td>
<td>260</td>
<td>213(81.92%)</td>
</tr>
<tr>
<td>PCI</td>
<td>51</td>
<td>467</td>
<td>351(75.16%)</td>
</tr>
<tr>
<td>USB</td>
<td>268</td>
<td>4148</td>
<td>2971(71.62%)</td>
</tr>
<tr>
<td>Total</td>
<td>848</td>
<td>18176</td>
<td>13578(74.70%)</td>
</tr>
</tbody>
</table>
CHARACTERISTIC 2

- Few branches
  - There are few if branches in error handling code
- Driver study
  - 78% of error handling code is out of the if branches
  - Reason: fail-stop model

<table>
<thead>
<tr>
<th>Driver Class</th>
<th>Number</th>
<th>Error handling</th>
<th>Without Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless</td>
<td>116</td>
<td>3903</td>
<td>3111(79.71%)</td>
</tr>
<tr>
<td>Ethernet</td>
<td>219</td>
<td>2587</td>
<td>1941(75.03%)</td>
</tr>
<tr>
<td>Block</td>
<td>56</td>
<td>149</td>
<td>127(85.23%)</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>21</td>
<td>330</td>
<td>239(72.42%)</td>
</tr>
<tr>
<td>Clock</td>
<td>117</td>
<td>467</td>
<td>422(90.36%)</td>
</tr>
<tr>
<td>PCI</td>
<td>51</td>
<td>470</td>
<td>371(78.94%)</td>
</tr>
<tr>
<td>USB</td>
<td>268</td>
<td>701</td>
<td>493(70.32%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>848</strong></td>
<td><strong>8607</strong></td>
<td><strong>6704(77.89%)</strong></td>
</tr>
</tbody>
</table>
CHARACTERISTIC 3

- Check decision
  - To check whether an occasional error occurs, an *if* check is often used in the source code
  - The checked data can be function return values *(C1)* or common variables
CHARACTERISTIC USAGE

- Function return value trigger (C1)
  - Injecting faults into function return values can cover most error handling code

- Few branches (C2)
  - Injecting single fault in each test case can cover most error handling code

- Check decision (C3)
  - The function whose return value is checked in the code should be fault-injected
EH-TEST

- Architecture
  - Fault extractor
  - Fault injector
  - Probe inserter
  - Runtime monitor
  - Pair checkers

- Two phases
  - Test case generation
  - Runtime testing
PHASE 1: TEST CASE GENERATION

Task 1: Extracting target functions

- Input: OS + driver source code
- Output: target functions
- Method: pattern-based extraction strategy

Diagram:

- Fault Extractor
- Fault Injector
- Probe Inserter
- Runtime Monitor
- Pair Checkers

Output:
- Target Functions
- Test Cases
- Bug Reports
PATTERN-BASED EXTRACTION

- Based on C1 and C3
- Three code patterns
- Automated and accurate extraction

Collect traces:

Simple extraction:
(candidates functions)

<table>
<thead>
<tr>
<th>Pattern 1:</th>
<th>Pattern 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern 3:</td>
<td></td>
</tr>
</tbody>
</table>

Procedure: Pattern-based extraction strategy

1. func_set := ∅; cand_set := ∅; fault_set := ∅;
2. func_set := called functions in normal execution traces;
3. foreach func in func_set do
   4. if GetRetType(func) == integer or pointer then
      5. AddSet(cand_set, func);
      6. end if
   7. end foreach
8. foreach func in cand_set do
   9. if func's RetVal is checked by "if" in the driver then
      10. AddSet(fault_set, func);
   11. else if func's RetVal is checked in other drivers then
      12. AddSet(fault_set, func);
   13. else if func's RetVal is specified to be checked then
      14. AddSet(fault_set, func);
   15. end if
   16. end foreach
PHASE 1: TEST CASE GENERATION

Task 2: Injecting faults into target functions
- Input: driver code + target functions
- Output: processed driver LLVM bytecode
- Method: single fault injection, code instrumentation
PHASE 1: TEST CASE GENERATION

- Task 3: Inserting probes for runtime monitoring
  - Input: processed driver LLVM bytecode
  - Output: driver test cases (loadable drivers)
  - Method: code instrumentation
PHASE 2: RUNTIME TESTING

- Runtime monitoring
  - Record runtime information
  - Maintain a resource-usage list
  - Measuring code coverage
PHASE 2: RUNTIME TESTING

- Bug reporting
  - Driver crashes
  - Driver hangs
  - Resource-release omissions
EVALUATION

- 15 common Linux drivers (3.1.1 and 3.17.2)
  - 4 wireless drivers
  - 3 USB drivers
  - 8 Ethernet drivers

<table>
<thead>
<tr>
<th>Class</th>
<th>Driver</th>
<th>Hardware</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless</td>
<td>rtl8180</td>
<td>Realtek RTL8180L Wireless Controller</td>
<td>4.6K</td>
</tr>
<tr>
<td></td>
<td>b43</td>
<td>Broadcom BCM4322 Wireless Controller</td>
<td>57.5K</td>
</tr>
<tr>
<td></td>
<td>iwl4965</td>
<td>Intel 4965AGN Wireless Controller</td>
<td>29.1K</td>
</tr>
<tr>
<td></td>
<td>rt2800</td>
<td>Ralink RT3060 Wireless Controller</td>
<td>22.5K</td>
</tr>
<tr>
<td>USB</td>
<td>usb_storage</td>
<td>Kingston 4GB USB disk</td>
<td>7.6K</td>
</tr>
<tr>
<td></td>
<td>uhci_hcd</td>
<td>Intel USB UHCI Controller</td>
<td>7.2K</td>
</tr>
<tr>
<td></td>
<td>ehci_hcd</td>
<td>Intel USB2 EHCI Controller</td>
<td>11.2K</td>
</tr>
<tr>
<td>Ethernet</td>
<td>e100</td>
<td>Intel 82559 Ethernet Controller</td>
<td>3.2K</td>
</tr>
<tr>
<td></td>
<td>e1000e</td>
<td>Intel 82572EI Ethernet Controller</td>
<td>28.3K</td>
</tr>
<tr>
<td></td>
<td>igb</td>
<td>Intel 82575EB Ethernet Controller</td>
<td>24.9K</td>
</tr>
<tr>
<td></td>
<td>r8169</td>
<td>Realtek RTL8169 Ethernet Controller</td>
<td>7.4K</td>
</tr>
<tr>
<td></td>
<td>8139too</td>
<td>Realtek RTL8139D Ethernet Controller</td>
<td>2.7K</td>
</tr>
<tr>
<td></td>
<td>3c59x</td>
<td>3Com 3c905B Ethernet Controller</td>
<td>3.4K</td>
</tr>
<tr>
<td></td>
<td>sky2</td>
<td>Marvell 88E8056 Ethernet Controller</td>
<td>7.7K</td>
</tr>
<tr>
<td></td>
<td>ipg</td>
<td>ICPlus IP1000 Ethernet Controller</td>
<td>3.0K</td>
</tr>
</tbody>
</table>
EVALUATION

- Target function extraction
  - 76% of candidate functions are filtered out
  - 10% false positive rate
  - 86% of target functions are called in initialization

<table>
<thead>
<tr>
<th>Driver</th>
<th>Candidate</th>
<th>Target</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtl8180</td>
<td>39</td>
<td>18</td>
<td>17 (14)</td>
</tr>
<tr>
<td>b43</td>
<td>260</td>
<td>55</td>
<td>55 (46)</td>
</tr>
<tr>
<td>iwl1945</td>
<td>497</td>
<td>79</td>
<td>74 (64)</td>
</tr>
<tr>
<td>rt2800</td>
<td>185</td>
<td>65</td>
<td>57 (48)</td>
</tr>
<tr>
<td>usb_storage</td>
<td>60</td>
<td>20</td>
<td>15 (15)</td>
</tr>
<tr>
<td>uhci_hcd</td>
<td>120</td>
<td>24</td>
<td>19 (10)</td>
</tr>
<tr>
<td>ehci_hcd</td>
<td>160</td>
<td>23</td>
<td>21 (14)</td>
</tr>
<tr>
<td>e100</td>
<td>80</td>
<td>33</td>
<td>27 (26)</td>
</tr>
<tr>
<td>e1000e</td>
<td>175</td>
<td>62</td>
<td>56 (41)</td>
</tr>
<tr>
<td>igb</td>
<td>247</td>
<td>59</td>
<td>51 (51)</td>
</tr>
<tr>
<td>r8169</td>
<td>77</td>
<td>15</td>
<td>15 (14)</td>
</tr>
<tr>
<td>8139too</td>
<td>64</td>
<td>9</td>
<td>8 (7)</td>
</tr>
<tr>
<td>3c59x</td>
<td>59</td>
<td>15</td>
<td>14 (14)</td>
</tr>
<tr>
<td>sky2</td>
<td>86</td>
<td>30</td>
<td>25 (25)</td>
</tr>
<tr>
<td>ipg</td>
<td>74</td>
<td>16</td>
<td>16 (15)</td>
</tr>
<tr>
<td>Total</td>
<td>2183</td>
<td>523</td>
<td>470 (404)</td>
</tr>
</tbody>
</table>
EVALUATION

- Bug detection
  - 32 real bugs in 3.1.1, 50 real bugs in 3.17.2
  - 9 bugs in 3.1.1 have been fixed in 3.17.2
  - 17 patches are sent, and 15 of them are applied
  - Many resource-release omissions

<table>
<thead>
<tr>
<th>Driver</th>
<th>Linux 3.1.1</th>
<th>Linux 3.17.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test case</td>
<td>Time usage</td>
</tr>
<tr>
<td>rt8180</td>
<td>16</td>
<td>03:14</td>
</tr>
<tr>
<td>b43</td>
<td>62</td>
<td>23:57</td>
</tr>
<tr>
<td>iwl4965</td>
<td>100</td>
<td>36:42</td>
</tr>
<tr>
<td>rt2800</td>
<td>62</td>
<td>19:21</td>
</tr>
<tr>
<td>usb_storage</td>
<td>25</td>
<td>03:35</td>
</tr>
<tr>
<td>uhci_hcd</td>
<td>22</td>
<td>03:47</td>
</tr>
<tr>
<td>ehci_hcd</td>
<td>24</td>
<td>03:50</td>
</tr>
<tr>
<td>e100</td>
<td>33</td>
<td>03:02</td>
</tr>
<tr>
<td>e1000e</td>
<td>66</td>
<td>11:01</td>
</tr>
<tr>
<td>igb</td>
<td>62</td>
<td>10:09</td>
</tr>
<tr>
<td>r8169</td>
<td>15</td>
<td>01:24</td>
</tr>
<tr>
<td>8139teo</td>
<td>9</td>
<td>00:45</td>
</tr>
<tr>
<td>3c59x</td>
<td>18</td>
<td>01:26</td>
</tr>
<tr>
<td>sky2</td>
<td>26</td>
<td>01:43</td>
</tr>
<tr>
<td>ipg</td>
<td>17</td>
<td>01:16</td>
</tr>
<tr>
<td>Total</td>
<td>557</td>
<td>125:12</td>
</tr>
</tbody>
</table>
EVALUATION

- Code coverage
  - Improve 8.8% in driver initialization
  - Not all error handling code can be covered
ADFI VS EH-TEST

- ADFI [ISSTA ’15]
  - SFI testing for drivers
  - Injecting faults into target function return values
  - Detect crashes, hangs and memory leaks

- Differences
  - Target functions are manually selected
  - Injecting multiple faults into each test case

- Bug detection
  - Find the same number of bugs in e100 and r8169
  - 10 bugs in ehci_hcd found by EH-Test are omitted
LIMITATIONS

- Some error handling code is uncovered
  - Single fault injection
  - Only injecting faults into function return values
- Only default configuration is covered
CONCLUSION

- Driver code study and 3 useful characteristics
- Automated and accurate method: pattern-based extraction strategy
- Efficient SFI approach: EH-Test
- 50 real bugs in 15 Linux drivers
- Future work: cover more error handling code and configurations
Thanks!

Q & A