Recovering Device Drivers

Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy
University of Washington

Presenter: Hayun Lee
Embedded Software Lab.

Symposium on Operating Systems Design and Implementation (OSDI) 2004
2017.05.22
Contents

• Introduction
• Key Idea
• Shadow Drivers
• Evaluation
• Conclusion
Introduction

• Improving **reliability** is one of the greatest challenges for commodity OS.
  – System failures are commonplace and costly

• Most of system failures are caused by the OS’s **device drivers**.
  – 85% of Windows XP crashes
  – Linux drivers have 7x the bug rate of other kernel code

Preventing driver-induced failures \(\rightarrow\) Improves overall system reliability
Introduction

• Failure-isolation systems (Nooks)
  – Their previously proposed technique
  – The kernel unloads a failed driver and then restarts it
    • Prevent driver failures from corrupting the kernel itself

• BUT, restarting does not prevent applications crash
  – driver loses application state
  – applications receive erroneous results

They propose “Shadow Drivers”
Let’s assume a blind date company. (just assumption)
- The company does not care about the conversation between woman and man.
Let’s assume a blind date company. (just assumption)
- The company monitors the conversation between woman and man.
Let's assume a blind date company. (just assumption)
- When a man speaks dangerous words...
Key Idea - Solution

- Let’s assume a blind date company. (just assumption)
  - When a man returns to normal...
Device Drivers

- **Device Driver**
  - A kernel-mode software component that provides an interface between the OS and a hardware device

- **Driver Faults**
  - Deterministic failure
    - Triggered by a sequence of configuration or I/O requests
    - No generic recovery technique can recovery
  - Transient failure
    - Triggered by additional inputs from the device or the OS
  - Fail-stop failure
    - Detected and stopped by the system before any OS, device or application state is affected

Embedded Software Lab. @ SKKU
Shadow Drivers

- Shadow drivers hide driver failures from applications and the OS

- Two modes
  - Passive mode
    - Monitoring: monitor kernel-driver communication
  - Active mode
    - Recovery: restart driver after failure
    - Proxying: act as driver during recovery
Passive Mode

- Monitoring
  - Tracks requests made to the driver
  - Records configuration and driver parameters that kernel passes into the driver
  - Tracks all kernel objects that the driver allocated or received from the kernel
• **Active Mode**
  - On failure, the tap switches to active mode
  - Terminates all communication between the driver and kernel
  - Redirects all invocations to their corresponding interface in the shadow
Active Mode

• **Recovery**
  – Stop the failed driver
  – Reinitialize the driver from a clean state
  – Transfer relevant shadow driver state into the new driver

• **Proxying of Kernel Requests**
  – Respond with information that it has recorded
  – Silently drop the request
  – Queue the request for later processing
  – Block the request until the driver recovers
  – Report that the driver is busy
Recovery Subsystem

- All shadow drivers rely on three functions
  - Isolation service
  - Redirection mechanism
  - Object tracking service

- Uses Nooks
  - Protection domains
  - Proxies + Taps
  - Object table
Evaluation

• 4 key aspects of shadow drivers
  – Performance
  – Fault-Tolerance
  – Limitations
  – Code size

• Implementation & Test
  – Build 3 Linux shadow drivers (for 3 device driver classes)
  – Test them on 13 different Linux drivers

<table>
<thead>
<tr>
<th>Class</th>
<th>Driver</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>e1000</td>
<td>Intel Pro/1000 Gigabit Ethernet</td>
</tr>
<tr>
<td></td>
<td>pcnet32</td>
<td>AMD PCnet32 10/100 Ethernet</td>
</tr>
<tr>
<td></td>
<td>3c59x</td>
<td>3COM 3c509b 10/100 Ethernet</td>
</tr>
<tr>
<td></td>
<td>e100</td>
<td>Intel Pro/100 Ethernet</td>
</tr>
<tr>
<td></td>
<td>epic100</td>
<td>SMC EtherPower 10/100 Ethernet</td>
</tr>
<tr>
<td>Sound</td>
<td>audigy</td>
<td>SoundBlaster Audigy sound card</td>
</tr>
<tr>
<td></td>
<td>emu10k1</td>
<td>SoundBlaster Live! sound card</td>
</tr>
<tr>
<td></td>
<td>sb</td>
<td>SoundBlaster 16 sound card</td>
</tr>
<tr>
<td></td>
<td>es1371</td>
<td>Ensoniq sound card</td>
</tr>
<tr>
<td></td>
<td>cs4232</td>
<td>Crystal sound card</td>
</tr>
<tr>
<td></td>
<td>i810_audio</td>
<td>Intel 810 sound card</td>
</tr>
<tr>
<td>Storage</td>
<td>ide-disk</td>
<td>IDE disk</td>
</tr>
<tr>
<td></td>
<td>ide-cd</td>
<td>IDE CD-ROM</td>
</tr>
</tbody>
</table>
Evaluation

- There are the applications used for evaluating shadow drivers

<table>
<thead>
<tr>
<th>Device Driver</th>
<th>Application Activity</th>
</tr>
</thead>
</table>
| Sound (audigy driver)| - mp3 player (*zinf*) playing 128kb/s audio  
                        - audio recorder (*audacity*) recording from microphone  
                        - speech synthesizer (*festival*) reading a text file  
                        - strategy game (*Battle of Wesnoth*) |
| Network (e1000 driver)| - network send (*netperf*) over TCP/IP  
                        - network receive (*netperf*) over TCP/IP  
                        - network file transfer (*scp*) of a 1GB file  
                        - remote window manager (*vnc*)  
                        - network analyzer (*ethereal*) sniffing packets |
| Storage (ide-disk driver)| - compiler (*make/gcc*) compiling 788 C files  
                          - encoder (*LAME*) converting 90 MB file .wav to .mp3  
                          - database (*mySQL*) processing the *Wisconsin Benchmark* |
Performance

- What is the performance overhead of shadow drivers during normal, passive-mode operation?

Not attempt to conceal its failure
Performance

- What is the performance overhead of shadow drivers during normal, passive-mode operation?

- Impose relatively little performance overhead

- Not attempt to conceal its failure
Can applications that use a device driver continue to run even after the driver fails?

<table>
<thead>
<tr>
<th>Device Driver</th>
<th>Application Activity</th>
<th>Linux-Native</th>
<th>Linux-Nooks</th>
<th>Linux-SD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sound</strong> (audigy driver)</td>
<td>mp3 player</td>
<td>CRASH</td>
<td>MALFUNCTION</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>audio recorder</td>
<td>CRASH</td>
<td>MALFUNCTION</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>speech synthesizer</td>
<td>CRASH</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>strategy game</td>
<td>CRASH</td>
<td>MALFUNCTION</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Network</strong> (e1000 driver)</td>
<td>network file transfer</td>
<td>CRASH</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>remote window manager</td>
<td>CRASH</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>network analyzer</td>
<td>CRASH</td>
<td>MALFUNCTION</td>
<td>✓</td>
</tr>
<tr>
<td><strong>IDE</strong> (ide-disk driver)</td>
<td>compiler</td>
<td>CRASH</td>
<td>CRASH</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>encoder</td>
<td>CRASH</td>
<td>CRASH</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>database</td>
<td>CRASH</td>
<td>CRASH</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Fault-Tolerance

- Can applications that use a device driver continue to run even after the driver fails?

<table>
<thead>
<tr>
<th>Device Driver</th>
<th>Application Activity</th>
<th>Application Behavior</th>
<th>Linux-Native</th>
<th>Linux-Nooks</th>
<th>Linux-SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound (audiogy driver)</td>
<td>mp3 player, audio recorder</td>
<td>CRASH, CRASH</td>
<td>CRASH, CRASH</td>
<td>MALFUNCTION, MALFUNCTION</td>
<td>✓</td>
</tr>
<tr>
<td>Network (e1000 driver)</td>
<td>network file transfer, remote window manager, network analyzer</td>
<td>CRASH, CRASH</td>
<td>CRASH, ✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IDE (ide-disk driver)</td>
<td>compiler, encoder, database</td>
<td>CRASH, CRASH, CRASH</td>
<td>CRASH, CRASH, CRASH</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Keep applications running when a driver fails.
Limitations

• How reasonable is our assumption that driver failures are fail-stop?

![Fault Injection Outcomes Graph]

- Sound: 78% Detected, 44% Recovered
- Network: 96% Detected, 76% Recovered
- Storage: 38% Detected, 58% Recovered

Embedded Software Lab. @ SKKU
Limitations

- How reasonable is our assumption that driver failures are fail-stop?

![Fault Injection Outcomes]

Limited by a system’s ability to detect that a driver has failed
Code size

- How much code is required for shadow drivers and their supporting infrastructure?

<table>
<thead>
<tr>
<th>Driver Class</th>
<th>Shadow Driver Lines of Code</th>
<th>Device Driver Shadowed Lines of Code</th>
<th>Class Size # of Drivers</th>
<th>Class Size Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound</td>
<td>666</td>
<td>7,381 (audigy)</td>
<td>48</td>
<td>118,981</td>
</tr>
<tr>
<td>Network</td>
<td>198</td>
<td>13,577 (e1000)</td>
<td>190</td>
<td>264,500</td>
</tr>
<tr>
<td>Storage</td>
<td>321</td>
<td>5,358 (ide-disk)</td>
<td>8</td>
<td>29,000</td>
</tr>
</tbody>
</table>
Code size

- How much code is required for shadow drivers and their supporting infrastructure?

<table>
<thead>
<tr>
<th>Driver Class</th>
<th>Shadow Driver Lines of Code</th>
<th>Device Driver Shadowed Lines of Code</th>
<th>Class Size # of Drivers</th>
<th>Class Size Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound</td>
<td></td>
<td></td>
<td></td>
<td>118,981</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
<td></td>
<td>264,500</td>
</tr>
<tr>
<td>Storage</td>
<td>321</td>
<td>5,358 (ide-disk)</td>
<td>8</td>
<td>29,000</td>
</tr>
</tbody>
</table>

Implemented with a modest amount of code
Conclusion

• **Shadow Drivers**
  – Mask device driver failures both the OS and applications
  – Improve application reliability
  – Efficient and little performance degradation
  – Transparent (requiring no code changes to existing drivers)

• **Discussion**
  – Why only three driver classes?