Introduction

What happens when the device fails while this device driver code executes (drivers/net/3cs9x.c)?

```c
while (ioread16(ioaddr + Wn7_MasterStatus) & 0x8000) {
    Unreliable devices are a major source of system downtime. One study indicates that drivers designed with fault tolerance reduced reboot rates due to faulty hardware from 8% to 3%. In addition, transient device failures are common [1, 3, 6] and stem from the sources shown at right. OS driver design guidelines, shown at far right, outline how to harden drivers manually.
```

Hardware Device Failures

Driver Design Guidelines [4, 7, 5]

<table>
<thead>
<tr>
<th>Validation</th>
<th>Reporting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input validation</td>
<td>Report hardware failures</td>
</tr>
<tr>
<td>Unrepeatable reads</td>
<td>Recovery</td>
</tr>
<tr>
<td>DMA protection</td>
<td>Handle all failures</td>
</tr>
<tr>
<td>Data corruption</td>
<td>Cleanup properly</td>
</tr>
<tr>
<td>Timing</td>
<td>Conceal failure</td>
</tr>
<tr>
<td>Infinite polling</td>
<td>Do not crash</td>
</tr>
<tr>
<td>Stuck interrupts</td>
<td>Test drivers</td>
</tr>
<tr>
<td>Lost request</td>
<td>Wrap I/O memory access</td>
</tr>
<tr>
<td>Excessive delay</td>
<td>Unexpected events</td>
</tr>
</tbody>
</table>

Device Failures ⇒ Crashes

- Bit flip faults
- Stuck-at faults
- Wear out
- Insufficient burn in
- Bridging faults
- Firmware bugs
- Electromagnetic interference

Recommendations addressed by Carburizer are in black.

Carburizer

Carburizer automatically implements many driver design guidelines and provides runtime support to recover failed drivers. It relies on static code analysis and runtime support to harden drivers against device failure, and to report bugs to developers.

Hardening Drivers

Carburizer corrects or reports four categories of bugs in device drivers: infinite polling, invalid array accesses, invalid pointer uses, and calls to the kernel’s panic() routine.

- **Infinite Polling**: Driver waits indefinitely for hardware device register to change
- **Invalid array access**: Driver uses a potentially bad value read from the device as an array index
- **Invalid pointer use**: Driver dereferences a potentially-invalid pointer read from the device
- **Calls panic()**: Driver crashes the system

```c
if ((pas_model = pas_read[8xFF88])) {
    char temp[180];
    sprintf(temp, %s rev %d", pas_model_name[int] pas_model),
    pas_read[8x27B99]; ...
}
```

- Driver accesses invalid memory if the device malfunctions
- Carburizer calculates the array’s size and inserts a bounds check, and calls recovery on failure

```c
if (ioread16(ioaddr + Wn7_MasterStatus) & 0x8000) {
    unsigned long delta = (cpu/kHz)/2;
    unsigned long start = 0, cur = 0;
    timeout = rdstcll(start) + delta;
    reg_val = readmio + PHY_ACCESS)
    sys_report("", module_name, device_id_d); return -1;
}
```

- Driver already detects the timeout, but does not report it
- Carburizer automatically reports the timeout to the centralized fault management system

Reporting Hardware Failures

Transient hardware failures often precede permanent failures [6]. Reporting recoverable errors can allow proactive replacement of failure-prone devices. Carburizer locates driver code that detects hardware failures, and ensures the error is reported. If not, Carburizer inserts a reporting routine at the location of each detected hardware malfunction, including device timeouts and conditionally returning negative constants based on values read from the hardware.

```c
reg_val = rdstcll(start) + delta;
reg_val = readmio + PHY_ACCESS)
reg_val = readmio + PHY_ACCESS)
```

- Driver loops forever if the device malfunctions or is disconnected
- Carburizer inserts a timeout, and calls the recovery mechanism if the device malfunctions

Runtime Fault Tolerance

The Carburizer runtime recovery system is based on Shadow Drivers [8]. The runtime also recovers from stuck and missing interrupts.

Dynamic Polling

Normally, devices generate interrupts in response to requests made of the driver. If interrupt activity stops, but requests continue, the runtime polls the interrupt handler. The runtime uses the return value to determine whether the call was productive. The polling rate doubles if the call was productive; otherwise, it halves. The runtime corrects interrupt storms by disabling the IRQ line and using the same polling mechanism.

Driver Activity via Referenced Bits

The runtime uses the referenced bits on the pages containing the driver’s code to determine when driver requests are made. If referenced bits are set, the driver has been invoked, so an interrupt is expected. If the interrupt handler is not called, it may indicate a missing interrupt. The following flow chart outlines the polling algorithm.

Hardware-Dependence Bugs in Linux

```c
Number of Bugs
```

- Net
- SCSI
- Sound
- Video
- Other

Reported Timeouts and Incorrect Outputs

```c
Number of Bugs
```

- Net
- SCSI
- Sound
- Other

References