Is Linux Kernel Oops Useful Or Not?

Takeshi Yoshimura†, Hiroshi Yamada†*, Kenji Kono†*
†Keio University  *JST/CREST

October 7 2012
OS Kernel Crash

- OSes need to be highly available
  - Necessary for all apps to continue running
  - A kernel crash can lead to the outage of the entire apps

- Kernel crashes are difficult to be zero
  - Bugs inside Linux still exist [Palix et al. ASPLOS ’11]
  - Bugs are not always fixed soon
What Is Kernel Oops?

• A Linux behavior to handle detected bugs
  – e.g., in-kernel NULL deref
• Linux kills a faulty context’s proc in kernel oops
  – Linux attempts to avoid kernel crashes, called “panic”
• Linux continues to run on a compromised reliability
Why Can Kernel Ooops Be Useful?

• Linux can remain reliable after kernel oops if errors are confined in a kernel context
  – Shared kernel objects remain correct
  – Non-faulty procs can continue running correctly
  • *Without rebooting or any complex mechanisms*
Error Propagation Scope

• **Process-local** error
  - Propagates only within the kernel context of a proc
    • e.g., kernel stack, function-local data
  - Errors can be removed by killing a faulty proc

• **Kernel-global** error
  - Propagates to data shared among kernel contexts
    • e.g., kernel states, global data, heap data

![Diagram showing process-local and kernel-global error propagation](image)
Goal in This Work

G-1: Analyze Linux behavior to faults
   – How frequently does Linux invoke oops/panic?

G-2: Analyze error propagation scope in oops
   – Are kernel states corrupted after fault activation?

G-3: Estimate the Linux reliability after kernel oops
   – How frequently can Linux avoid panic correctly?

• Explore the possibility of using kernel oops as an error recovery method
Experimental Equipment

- Linux 2.6.38 kernel on VMware Workstation 8
  - 1 CPU, 1GB memory, 20GB Disk
- A fault injector used by existing work
  - [Ng et al. ’98], [Swift et al. ’03], [Depoutovitch et al. ’10]
  - Obtained from Nooks Research web site
    - http://nooks.cs.washington.edu/
- KDB, a kernel debugger
  - To trace error propagation
- Six benchmarks as workloads
  - UnixBench on {ext4, fat, USB}, Netperf, Aplay and restarting all the daemon
The Fault Injector

• Emulates 15 fault types
  – Mutates random instr in the running kernel text
  – Extended to imitate some reported bugs in [Palix et al. ASPLOS ’11]
    • e.g., deleting NULL check

Examples of the Injected Fault

<table>
<thead>
<tr>
<th>Fault type</th>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td><code>int x = 1;</code></td>
<td><code>int x;</code></td>
</tr>
<tr>
<td>irq</td>
<td><code>arch_local_irq_restore()</code></td>
<td>deleted.</td>
</tr>
<tr>
<td>off by one</td>
<td><code>while (x &lt; 10)</code></td>
<td><code>while (x &lt;= 10)</code></td>
</tr>
<tr>
<td>bcopy</td>
<td><code>memcpy(ptr, ptr2, 256);</code></td>
<td><code>memcpy(ptr, ptr2, 512);</code></td>
</tr>
<tr>
<td>size</td>
<td><code>ptr = kmalloc(256, GFP_KERNEL);</code></td>
<td><code>ptr = kmalloc(128, GFP_KERNEL);</code></td>
</tr>
<tr>
<td>free</td>
<td><code>kfree(ptr);</code></td>
<td>deleted.</td>
</tr>
<tr>
<td>null</td>
<td><code>if (ptr == NULL) return;</code></td>
<td>deleted.</td>
</tr>
</tbody>
</table>
G-1: Analyzing Linux behavior

- Inject a fault
- Set a breakpoint to the faulty instr
- Run every workload in 6 benchmarks
- See if the fault is activated
  - If the kernel hits the breakpoint
- See what happens until the workload fails
  - Or until the workload is finished
G-1 Result: Failure By Fault Type

- 887 faults are activated (6738 are injected)
  - 75%: not manifested
  - 15%: oops, panic (propagation scope is investigated)
  - 10%: fail silence violation, hang, terminated by VMM
G-2: Analyzing Error Propagation

• Inject a fault causing kernel oops/panic
• Set a breakpoint to the faulty instr
• Run a workload
  – Wait until the kernel hits the breakpoint
• Trace instrs until the kernel oops
  – Currently, examine if stack or heap is corrupted
  – Analysis similar to a taint-analysis
G-2 Result: Scope Analysis

- 124 kernel oops & 10 panic are investigated
  - 73%: process-local error
  - 27%: kernel-global error
- Overrun, corrupt list_head or callback ptr, etc.

![Bar chart showing proc-local and kern-global errors for various categories such as branch, inverse, ptr, dstsrc, init, irq, off-by-one, size, bcopy, loop, var, null, and total. The chart visually represents the percentage of proc-local and kern-global errors across different categories.](image)
G-3: Estimating Reliability

- Inject a fault
- Run a workload
- Confirm kernel oops and the kernel kills a proc
- Remove the injected fault by using KDB
  - To imitate transient faults by the existing injector
- Run a workload in 6 benchmarks for each oops
- See what happens until the workload fails
  - Or until the workload is finished
G-3 Result: Failure After Oops By Scope

• 589 workloads are investigated
  – 58.7% of the workloads keep running
    • Workloads use a subsystem unrelated to the error
  – 40.8% of the workloads stop or do not start
    • Deadlock, oops/panic, and killing a important proc
  – 0.5% of the workloads run incorrectly

![Graph showing the distribution of failure after oops by scope](image-url)
Is Linux Kernel Oops Useful?

- 99.5% of the workloads run correctly or fail-stop after kernel oops
  - Deadlock occurs context’s fail-stop
    - The mutual execution is done to write shared data
    - A context killed in a critical section holds the lock
  - Linux shows fail-stopness even when errors are kernel-global
Related work

• A study of Linux behavior under errors [Gu et al. DSN ’03]
  – Conduct fault injection experiments
  – Show error propagation among subsystems

• Linux faults study [Palix et al. ASPLOS ’11]
  – Use a static analyzer to Linux kernels
  – Show the life-time and the distribution of bugs in Linux

• Reboot-based recovery with apps’ state reserved
  [Depoutovitch et al. EuroSys ’10, HotDep ’08]
  – Switch to the slave kernel when the master kernel crashes
  – Take downtime & need to re-design apps
Conclusion

• OSes need to be highly available
• Linux kills only a faulty proc instead of crashes
  – This kernel behavior is called “kernel oops”
  – Any complex mechanisms are not required
• Kernel oops can be useful as an error recovery
  – 99.5% of workloads run correctly or fail-stop after killing a faulty process