Guest operating system debugging

Find out what's wrong and what's right.

David Hildenbrand, Software Engineer Virtualization and Linux Development
19. August 2015, KVM Forum 2015
Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

Brian W. Kernighan and P. J. Plauger in *The Elements of Programming Style*.

Bugs – are they too strong? You're too weak!

Jeffrey Jedele (IBM employee)
Agenda

- Why debug guests?
- How bugs make your life hard
- Debugging techniques
- Advanced use cases
- Usage examples
- Outlook
- (Tips and Tricks)
Why debug guests?

- **Fix bugs** in a guest virtualization specific driver
- **Fix bugs** in the guest kernel
- **Fix bugs** in the BIOS / bootloader
- **Fix bugs** in the VMM by observing the effects on the guest
- See how the code works in „real life“
  - -> Understand the system, *avoid bugs*
- **Gain serious kernel hacking cred ;)**
How bugs make your life hard

- **Crashes**
  - Unrecoverable
- **Performance degradation**
  - E.g. due to inefficient locking, polling ...
  - System stays alive but is slow
- **Incorrect behaviour**
  - System stays alive but doesn't behave as expected
- **Deadlocks**
  - System might stay alive if it's not in the core
  - May be hard to reproduce
- **Data Corruption**
  - E.g. from random memory overwrites
  - System might stay alive if it's not in the core
  - May be hard to reproduce

As given in „Linux Kernel Development“ by Robert Love
Debugging Techniques (1) – Three approaches

Focus of this presentation
## Debugging Techniques (2) – Overview

<table>
<thead>
<tr>
<th></th>
<th>Guest (in)</th>
<th>VMM (out)</th>
<th>Guest (out)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logging</strong></td>
<td>Printk, debugfs, ...</td>
<td>Printf, logfiles, -d (tcg only) ...</td>
<td><em>(via guest memory)</em></td>
</tr>
<tr>
<td><strong>Tracing</strong></td>
<td>KGTP, strace, dtrace ..</td>
<td>e.g. qemu + kvm traces</td>
<td><em>(via gdb scripts)</em></td>
</tr>
<tr>
<td><strong>Dumps</strong></td>
<td>kexec/kdump + crash/gdb</td>
<td>e.g. process dump + gdb</td>
<td>QEMU guest dump</td>
</tr>
<tr>
<td><strong>Profiling</strong></td>
<td>oprofile/perf</td>
<td>oprofile/perf</td>
<td><em>perf kvm</em></td>
</tr>
<tr>
<td><strong>System Utilities</strong></td>
<td>top, /proc, /sys ...</td>
<td><em>perf kvm stat</em>, QEMU monitor</td>
<td>QEMU monitor</td>
</tr>
<tr>
<td><strong>Interactive Debugger</strong></td>
<td>KDB, KGDB, crash/gdb + /proc/kcore</td>
<td>gdb</td>
<td><em>gdbserver in QEMU</em></td>
</tr>
</tbody>
</table>

reuse to debug the host
Debugging Techniques (3) - Problems with inbound techniques

SELF -

Image source: http://kpc.am/1dZpT6f
Debugging Techniques (4) - Problems with inbound techniques

- A *(minimum) functional system* is required (kexec ready and working)
- Availability and quality depends on guest OS
- *Not all information accessible* (or very hard to get / decompose)
  - Early boot code
  - Interrupt handlers
- *Restricted to guest OS* (bootloader, (pc)bios)
- *Not transparent* to the guest
  - Guest might behave differently when active
- Most have to be *enabled/configured/installed before lightning strikes*

-> **Still very useful for many debugging scenarios**
Debugging Techniques (5) – perf kvm

**E.g. perf kvm --host --guest --guestvmlinux=/boot/vmlinux-custom --guestkallsyms=kallsyms top -e cpu-clock**

<table>
<thead>
<tr>
<th>Samples: 834K of event 'cpu-clock', Event count (approx.): 55230587977</th>
<th>Overhead</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>75.59% [kernel]</td>
<td>8,91%</td>
<td>[g] system_call</td>
<td>[k] enabled_wait</td>
</tr>
<tr>
<td>2,89% [guest.kernel]</td>
<td>1,93%</td>
<td>[g] fsnotify</td>
<td>[g] clear_user</td>
</tr>
<tr>
<td>1,17% [guest.kernel]</td>
<td>1,10%</td>
<td>[g] __fsnotify_parent</td>
<td>[g] security_file_permission</td>
</tr>
<tr>
<td>0,94% [guest.kernel]</td>
<td>0,94%</td>
<td>[g] vfs_write</td>
<td>[g] common_file_perm</td>
</tr>
<tr>
<td>0,86% [guest.kernel]</td>
<td>0,77%</td>
<td>[g] rw_verify_area</td>
<td>[g] __fget_light</td>
</tr>
<tr>
<td>0,74% [guest.kernel]</td>
<td>0,67%</td>
<td>[g] vfs_read</td>
<td>[g] __vfs_read</td>
</tr>
<tr>
<td>0,59% [guest.kernel]</td>
<td></td>
<td>[g] iov_iter_zero</td>
<td></td>
</tr>
</tbody>
</table>

**Press '?' for help on key bindings**

**system_call /boot/vmlinux-custom**

- ENTRY(system_call)
- stpt 688
- .Lsysc_stmg:
  - stmg %r8,%r15,__LC_SAVE_AREA_SYNC
  - lg %r10,___LC_LAST_BREAK
  - lg %r12,__LC_THREAD_INFO
  - lg %r12,792
  - lgh %r14,_PIF_SYSCALL
  - lghi %r14,1

**Press 'n' for help on key bindings**

### Diagram:

- **Take CPU sample**
  - (event trigger)
  - (was in hw virt)

- **Take VCPU sample**
  - (was not in hw virt)

- **Add as host sample**

- **Add as guest sample**
Debugging Techniques (6) – gdbserver in QEMU

- With KVM, hardware support is required for single-stepping, break-/watchpoints
- No extra disk space needed (in contrast to dumps)
- Remote GDB side „tracing” possible but slow
- Kernel with debug symbols only in remote GDB required
Debugging Techniques (7) - which outbound technique might help?

- **Crashes?**
  - QEMU dump, QEMU monitor or interactive debugging (“big guests“)
- **Performance degredation?**
  - `perf kvm stat / perf kvm`
  - Interactive debugging / guest tracing (after finding the hot spots)
- **Incorrect behaviour?**
  - Interactive debugging, guest tracing
- **Deadlocks?**
  - Interactive debugging (esp. pause/step single threads/vcpus)
  - Guest tracing
- **Data Corruption?**
  - Interactive debugging (esp. Watchpoints), guest tracing
Advanced use cases

- Understand and *fix bug reports without hardware at hand*
  - „VM should behave like real hardware“ (emulated devices)
- Debug scenarios that can *barely be seen in real life*
  - Simulate and debug device error conditions
  - E.g. on z Systems simulate cpu or device failures (TBD)
- What happens if ... *simulate bugs*
  - E.g. overwrite return values from functions
    - -> see how the system reacts (e.g. driver failure)
- Debug *software for hardware that is not available yet*
  - E.g. new hardware bringup (requires hw emulation)
Usage example (1): facility bug in early boot code

- No output, no error indication except bad PSW on KVM
- `qemu-system-s390x -s -S -kernel /boot/vmlinux ...
- `gdb /boot/vmlinux -tui -ex "target remote localhost:1234" -d ~/linux/

Analyze, single-step, break, modify ...
Usage example (1): facility bug in early boot code

- Early boot check for required facilities tested for a wrong one
- Current hardware typically has both facilities, **KVM did not**
  -> Bug triggered only in KVM (not on test systems)

commit 4a36b44c77515calad799577d3f9e2fa4d68bffa
Author: David Hildenbrand <dahi@linux.vnet.ibm.com>
Date: Wed Jun 18 12:32:19 2014 +0200

s390: require mvicos facility, not tod clock steering facility

```
#if defined(CONFIG_64BIT)
#elif defined(CONFIG_MARCH_ZEC12)
-  .long 3, 0xc100efe, 0xf46ce800, 0x004000000
+  .long 3, 0xc100efe, 0xf46ce800, 0x004000000
#endif

#elif defined(CONFIG_MARCH_Z196)
-  .long 2, 0xc100efe, 0xf46c0000
+  .long 2, 0xc100efe, 0xf46c0000
#endif

#elif defined(CONFIG_MARCH_Z10)
-  .long 2, 0xc100efe, 0xf0680000
+  .long 2, 0xc100efe, 0xf0680000
#endif

#elif defined(CONFIG_MARCH_Z9_109)
  .long 1, 0xc100efc2
```
Usage example (2): diag 44 in cpu_relax()

- **Performance regression** on new kernels
  - Only visible on CPU overcommitment, many vcpus
  - Long boot times, module loading extremely slow
- *e.g. perf kvm state live -d 10*
  - Run same workload on old and new kernel
  - Compare VM-EXIT / intercept results

<table>
<thead>
<tr>
<th>VM-EXIT</th>
<th>Samples</th>
<th>Samples%</th>
<th>Time%</th>
<th>Min Time</th>
<th>Max Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait state</td>
<td>8823</td>
<td>29.95%</td>
<td>99.36%</td>
<td>0.51us</td>
<td>4984120.42us</td>
</tr>
<tr>
<td>DIAG (0x44) time slice end</td>
<td>5884</td>
<td>19.97%</td>
<td>0.02%</td>
<td>0.90us</td>
<td>232.57us</td>
</tr>
<tr>
<td>SIGP emergency signal</td>
<td>5642</td>
<td>19.15%</td>
<td>0.03%</td>
<td>1.21us</td>
<td>1162.31us</td>
</tr>
<tr>
<td>Host interruption</td>
<td>4053</td>
<td>13.76%</td>
<td>0.02%</td>
<td>0.33us</td>
<td>2145.47us</td>
</tr>
<tr>
<td>DIAG (0x9c) time slice end directed</td>
<td>2624</td>
<td>8.91%</td>
<td>0.01%</td>
<td>0.94us</td>
<td>112.85us</td>
</tr>
<tr>
<td>DIAG (0x500) KVM virtio functions</td>
<td>1477</td>
<td>5.01%</td>
<td>0.01%</td>
<td>1.01us</td>
<td>158.75us</td>
</tr>
<tr>
<td>Partial-execution</td>
<td>290</td>
<td>0.98%</td>
<td>0.00%</td>
<td>0.40us</td>
<td>12.88us</td>
</tr>
<tr>
<td>0xB2 SERVC</td>
<td>178</td>
<td>0.60%</td>
<td>0.02%</td>
<td>17.48us</td>
<td>5876.41us</td>
</tr>
<tr>
<td>I/0 request</td>
<td>168</td>
<td>0.57%</td>
<td>0.00%</td>
<td>0.35us</td>
<td>13.52us</td>
</tr>
<tr>
<td>External request</td>
<td>79</td>
<td>0.27%</td>
<td>0.00%</td>
<td>0.41us</td>
<td>552.83us</td>
</tr>
<tr>
<td>0xB2 STSCH</td>
<td>79</td>
<td>0.27%</td>
<td>0.00%</td>
<td>4.17us</td>
<td>20.09us</td>
</tr>
<tr>
<td>SIGP</td>
<td>29</td>
<td>0.10%</td>
<td>0.00%</td>
<td>14.62us</td>
<td>103.66us</td>
</tr>
<tr>
<td>0xB2 SSCH</td>
<td>22</td>
<td>0.07%</td>
<td>0.00%</td>
<td>7.58us</td>
<td>178.29us</td>
</tr>
<tr>
<td>0xB2 TSCH</td>
<td>22</td>
<td>0.07%</td>
<td>0.00%</td>
<td>5.92us</td>
<td>41.46us</td>
</tr>
<tr>
<td>0xB2 STSI</td>
<td>13</td>
<td>0.04%</td>
<td>0.00%</td>
<td>0.76us</td>
<td>31.82us</td>
</tr>
</tbody>
</table>
Usage example (2): diag 44 in cpu_relax()

- „diag 44“ intercept == voluntarily give up time slice
  - Number drastically changed
- All VCPUs waiting for all VCPUs in stop_machine()
  - All VCPUs have to be scheduled once by the hypervisor
  - If VCPUs hand of time slices (diag 44), this happens much faster

```c
void cpu_relax(void) {
  barrier();
}
+void cpu_relax(void) {
  +if (!smp_cpu_mtid && MACHINE_HAS_DIAG44) {
    +asm volatile("diag 0,0,0x44");
    +barrier();
  }
  +EXPORT_SYMBOL(cpu_relax);
  +}
```

commit 4d92f50249eb3ed1c066276e214e8cc7be81e96d
Author: Heiko Carstens <heiko.carstens@de.ibm.com>
Date: Wed Jan 28 07:43:56 2015 +0100

s390: reintroduce diag 44 calls for cpu_relax()
Outlook

- Guest tracing
  - QEMU gdbserver support missing (see Google Summer of Code)
  - Requires at least support for single-stepping + breakpoints
  - HW support?
- „Live crash tool“
  - Attach crash to a living remote target (QEMU's gdbserver)
  - Convert crash features into gdb (python) scripts
- Support for more architectures + more hw support
  - HW debugging: x86, s390x, powerpc supported – arm tbd
- Allow to simulate more hardware varieties
  - E.g. CPU models on z Systems
  - Expose more „fake“ registers via QEMU's gdbserver
  - e.g. z Systems „last_break“ -> „where did I come from“

http://wiki.qemu.org/Google_Summer_of_Code_2012#Tracepoint_support_for_the_gdbstub
Tips and Tricks (1)

- Ways to start the QEMU gdbserver
  - -s: Start it directly (can also be passed using libvirt)
  - -s -S: Start it, don't start the guest (continue using gdb or QEMU monitor)
- Lazily using the QEMU monitor (gdbserver)
- Access the QEMU monitor using GDB „monitor“ command
  -> QEMU monitor access when using libvirt possible
- Debug binaries without debugging symbols
  - Architecture not announced via GDB remote protocol yet
  - Use e.g. "set arch s390:64-bit"
- Python bindings for GDB are really powerful
  - E.g. connect two GDBs to verify on breakpoint level (e.g. between QEMUs)
- Debug loadable kernel modules
  - getsyms.sh from kgdb
- Gdb scripts to be used in the remote GDB
  - Linux kernel: Documentation/kdump/gdbmacros.txt
Tips and Tricks (2): debug pcbios <-> kernel transition

boot / ipl

(load kernel from boot device into ram)

pcbios (a.k.a s390-ccw)

guest kernel

reboot / reipl

(reload bios into ram)

bootindex (initial boot device)

Debug pcbios code

Debug kernel code

(chreipl)

(change boot device)
Tips and Tricks (2): debug pcbios <-> kernel transition

- Both code parts *lie in guest memory* and *don't overlap*
  - pcbios overwrites kernel, kernel might overwrite pcbios
- Start qemu with the freshly compiled bios
  
  ```
  qemu-system-s390x -s -S -bios ~/pcbios/s390-ccw/s390-ccw.elf ...
  ```
- Start the remote gdb with the kernel, specify both source dirs
  
  ```
  gdb /boot/vmlinux -tui -ex "target remote localhost:1234 -d ~/linux/
  -d ~/qemu/
  ```
- Tell gdb about the pcbios (symbols + loaded location)
  
  ```
  add-symbol-file qemu/pc-bios/s390-ccw/s390-ccw.elf 0x3FE00400
  ```
- Use hw breakpoints (reloading overwrites sw breakpoints)
  
  ```
  hbreak jump_to_IPL_2 // e.g. just before starting kernel code
  hbreak *0x10014 // depends on kernel code
  ```

*(depends on qemu version, memory size and s390-ccw.elf, contact me for a calculation script)*
Tips and Tricks (2): just before the transition

```c
static ResetInfo save;

static void jump_to_IPL_2(void)
{
    ResetInfo *current = 0;

    void (*ipl)(void) = (void *) (uint64_t) current->ipl_continue;
    *current = save;

    ipl(); /* should not return */
}

static void jump_to_IPL_code(uint64_t address)
{
    /* store the subsystem information _after_ the bootmap was loaded */
    writeSubsystemIdentification();
}
```

Program received signal SIGTRAP, Trace/breakpoint trap.
jump_to_IPL_2 () at bootstrap.c:42
(gdb) step
(gdb) x current->ipl_continue
0xa050 <.lowcase+38918>: 0x000004170
(gdb) step
(gdb) 

Tips and Tricks (2): after the transition

```
arch/s390/kernel/head.S

358   j      .Lep_startup_kdump
359   .Lep_startup_normal:

> 360     mvi   LC_AR_MODE_ID,1   # set esame flag
361     slr   %r0,%r0           # set cpuid to zero
362     lhi   %r1,2            # mode 2 = esame (dump)
363     sigp  %r1,%r0,0x12      # switch to esame mode
364     bras  %r13,0f          #
365     .fill   16,4,0x0       #
366     0:     lmh   %r0,%r15,0(%r13) # clear high-order half of gprs
367     sam31  # switch to 31 bit addressing m
368     basr  %r13,0           # get base
369   .LPG0:
370     xc    0x200(256),0x200 # partially clear lowcore
371     xc    0x300(256),0x300
372     xc    0xe00(256),0xe00
```

remote Thread 1 In: startup_kdump L360 PC: 0x10014

(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
startup_kdump () at arch/s390/kernel/head.S:360
(gdb) layout prev
(gdb)
Thank you!

david.hildenbrand@de.ibm.com
# Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

<table>
<thead>
<tr>
<th>BlueMix</th>
<th>ECKD</th>
<th>IBM*</th>
<th>Maximo*</th>
<th>Smarter Cities*</th>
<th>WebSphere*</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInsights</td>
<td>FICON*</td>
<td>ibm.com</td>
<td>MQSeries*</td>
<td>Smarter Analytics</td>
<td>XIV*</td>
</tr>
<tr>
<td>Cognos*</td>
<td>FileNet*</td>
<td>IBM (logo)*</td>
<td>Performance Toolkit for VM</td>
<td>SPSS*</td>
<td>z13</td>
</tr>
<tr>
<td>DB2*</td>
<td>FlashSystem</td>
<td>IMS</td>
<td>POWER*</td>
<td>Storwize*</td>
<td>zEnterprise*</td>
</tr>
<tr>
<td>DB2 Connect</td>
<td>GDPS*</td>
<td>Informix*</td>
<td>Quickr*</td>
<td>System Storage*</td>
<td>z/OS*</td>
</tr>
<tr>
<td>Domino*</td>
<td>GPFS</td>
<td>InfoSphere</td>
<td>Rational*</td>
<td>Tivoli*</td>
<td></td>
</tr>
<tr>
<td>DS8000*</td>
<td></td>
<td></td>
<td>Sametime*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Windows Server and the Windows logo are trademarks of the Microsoft group of countries.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

* Other product and service names might be trademarks of IBM or other companies.