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)*

©2015 IBM Corporation

Why debug guests?

- *Fix bugs* in a guest virtualization specific driver
- *Fix bugs* in the 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* ;)

©2015 IBM Corporation

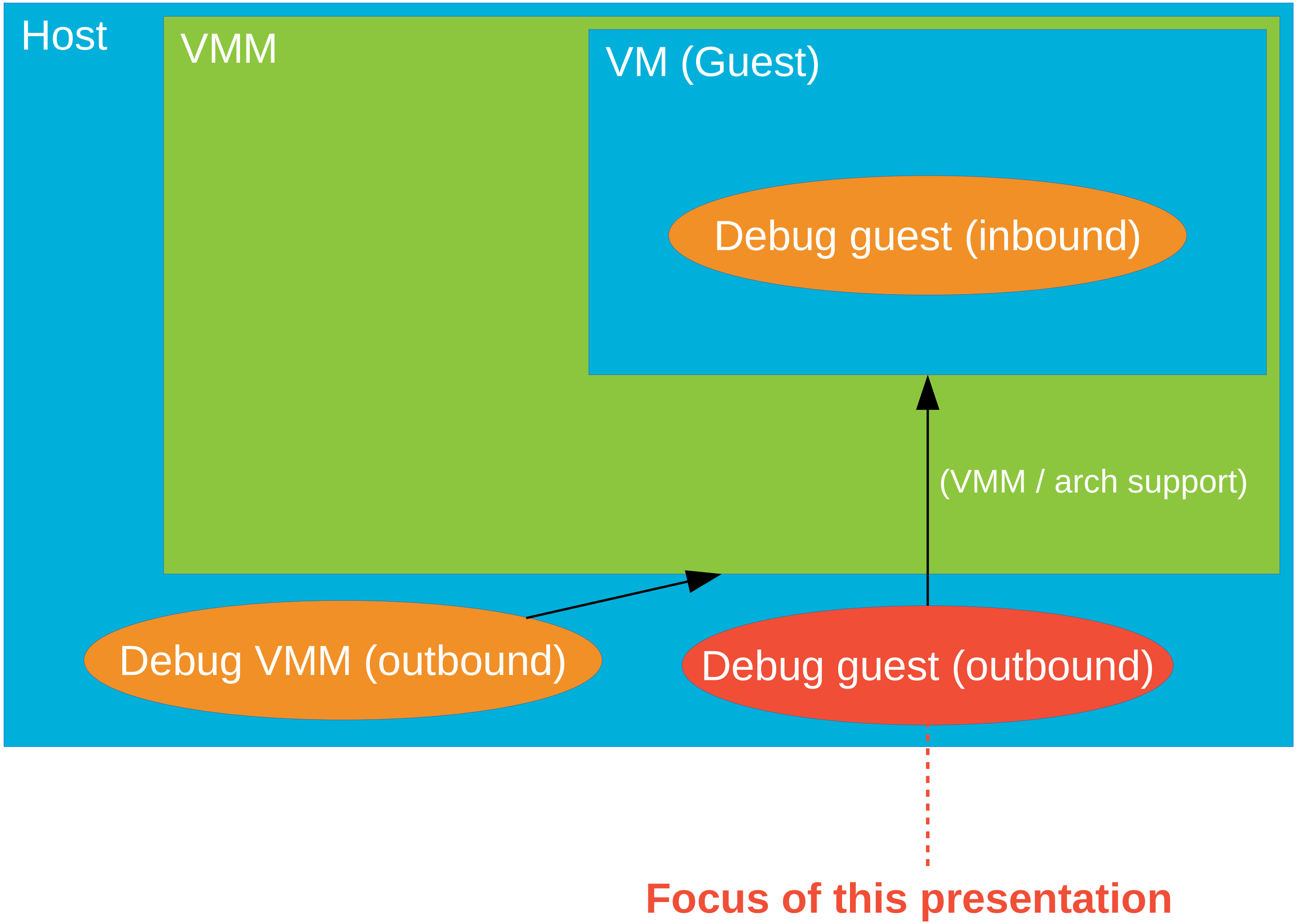
How bugs make your life hard

- *Crashes*
  - Unrecoverable
- *Performance degredation*
  - 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

©2015 IBM Corporation

Debugging Techniques (1) – Three approaches



©2015 IBM Corporation

Debugging Techniques (2) – Overview

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

reuse to debug the host

©2015 IBM Corporation

Debugging Techniques (3) - Problems with inbound techniques

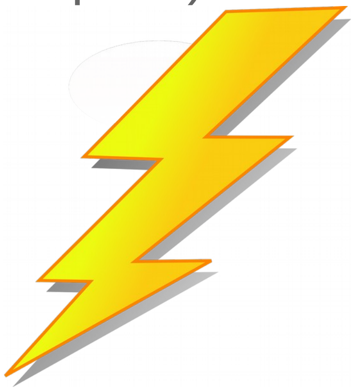
SELF -

Image source : http://kpc.am/1dZpT6f

 ©2015 IBM Corporation

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 usefull for many debugging scenarios

Image source: https://pixabay.com/p-297580/?no_redirect

©2015 IBM Corporation

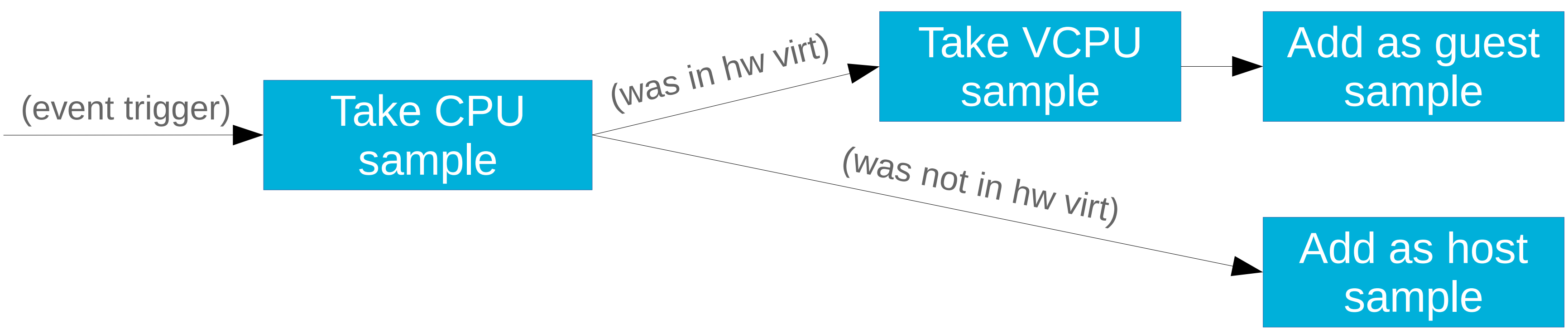
Debugging Techniques (5) – perf kvm

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

```
Samples: 834K of event 'cpu-clock', Event count (approx.): 55230587977
Overhead  Shared Object                Symbol
  75,59%  [kernel]                     [k] enabled_wait
   8,91%  [guest.kernel]               [g] system_call
   2,89%  [guest.kernel]               [g] fsnotify
   1,93%  [guest.kernel]               [g] __clear_user
   1,17%  [guest.kernel]               [g] __fsnotify_parent
   1,10%  [guest.kernel]               [g] security_file_permission
   0,94%  [guest.kernel]               [g] vfs_write
   0,94%  [guest.kernel]               [g] common_file_perm
   0,86%  [guest.kernel]               [g] rw_verify_area
   0,77%  [guest.kernel]               [g] __fget_light
   0,74%  [guest.kernel]               [g] vfs_read
   0,67%  [guest.kernel]               [g] __vfs_read
   0,59%  [guest.kernel]               [g] iov_iter_zero
Press '?' for help on key bindings
```

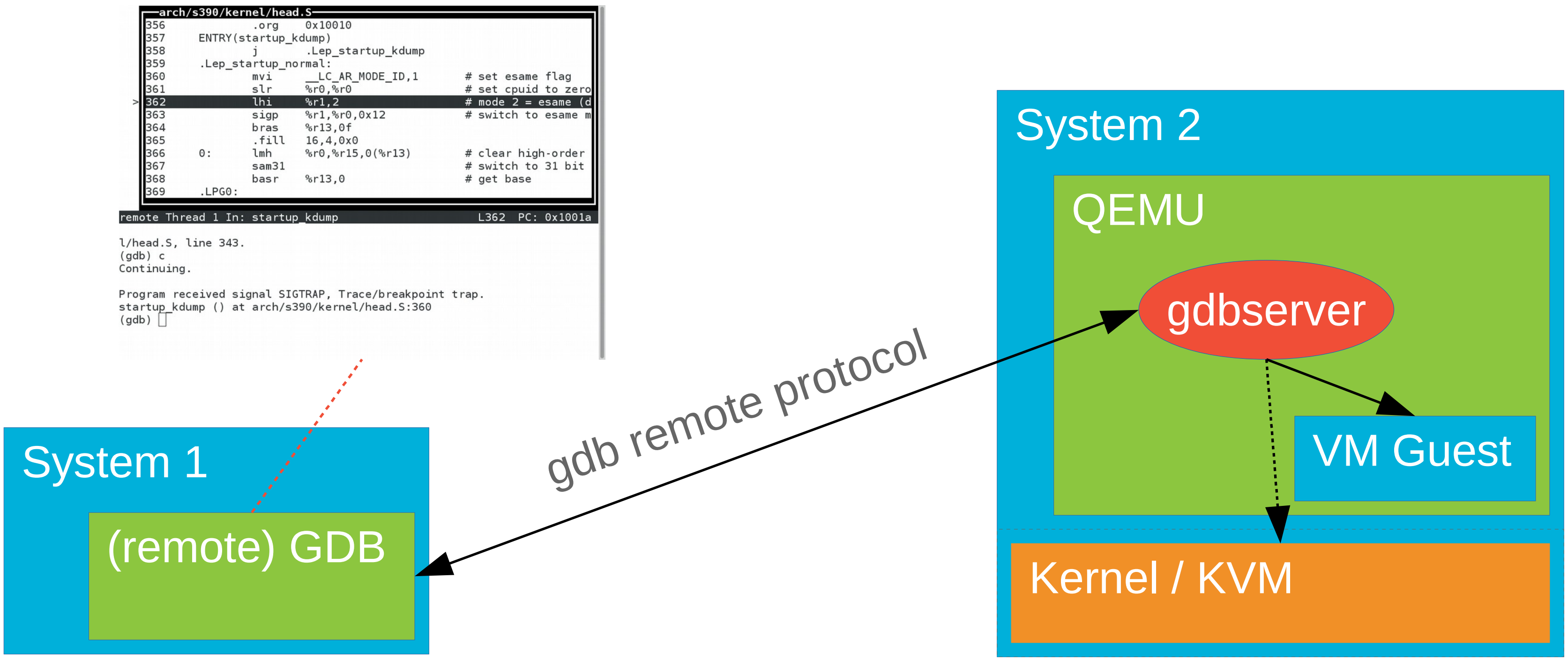
```
system_call  /boot/vmlinux-custom

             ENTRY(system_call)
                     stpt    __LC_SYNC_ENTER_TIMER
 65,47         stpt   688
             .Lsysc_stmg:
                     stmg    %r8,%r15,__LC_SAVE_AREA_SYNC
  4,91         stmg   %r8,%r15,512
                     lg      %r10,__LC_LAST_BREAK
  1,33         lg     %r10,272
                     lg      %r12,__LC_THREAD_INFO
               lg     %r12,792
                     lghi    %r14,_PIF_SYSCALL
  0,29         lghi   %r14,1
             .Lsysc_per:
Press 'h' for help on key bindings
```



 ©2015 IBM Corporation

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

©2015 IBM Corporation

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

©2015 IBM Corporation

Advanced use cases

developer's choice

- 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)

©2015 IBM Corporation

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/*

```
   ┌─arch/s390/kernel/head.S──────────────────────────────────────────────────
   │382            .insn   s,0xb2b00000,__LC_STFL_FAC_LIST # store facility list extended
   │383            # verify if all required facilities are supported by the machine
   │384     0:     la      %r1,__LC_STFL_FAC_LIST
   │385            la      %r2,3f+8-.LPG0(%r13)
   │386            l       %r3,0(%r2)
   │387     1:     l       %r0,0(%r1)
   │388            n       %r0,4(%r2)
   │389            cl      %r0,4(%r2)
B+>│390            jne     2f
   │391            la      %r1,4(%r1)
   │392            la      %r2,4(%r2)
   │393            ahi     %r3,-1
   │394            jnz     1b
   │395            j       4f
b+ │396     2:     l       %r15,.Lstack-.LPG0(%r13)
   │397            ahi     %r15,-96
   │398            la      %r2,.Lals_string-.LPG0(%r13)
   │399            l       %r3,.Lsclp_print-.LPG0(%r13)
remote Thread 1 In: startup_kdump                                        L390  PC: 0x100c2
(gdb) p /x $r1
$4 = 0xc8
(gdb) p /x $r2
$5 = 0x10148
(gdb) p /x $cc
$6 = 0x1
(gdb) set $cc=0
(gdb) p /x $cc
$7 = 0x0
(gdb)
```

Analyze, single-step, break, modify ...

 ©2015 IBM Corporation

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 4a36b44c77515ca1ad799577d3f9e2fa4d68bffa
Author: David Hildenbrand <dahi@linux.vnet.ibm.com>
Date:   Wed Jun 18 12:32:19 2014 +0200

    s390: require mvcos facility, not tod clock steering facility
```

```
 #if defined(CONFIG_64BIT)
 #if defined(CONFIG_MARCH_ZEC12)
-    .long 3, 0xc100efea, 0xf46ce800, 0x00400000
+ .long 3, 0xc100eff2, 0xf46ce800, 0x00400000
 #elif defined(CONFIG_MARCH_Z196)
-    .long 2, 0xc100efea, 0xf46c0000
+ .long 2, 0xc100eff2, 0xf46c0000
 #elif defined(CONFIG_MARCH_Z10)
-    .long 2, 0xc100efea, 0xf0680000
+ .long 2, 0xc100eff2, 0xf0680000
 #elif defined(CONFIG_MARCH_Z9_109)
        .long 1, 0xc100efc2
```

 ©2015 IBM Corporation

Usage example (2): diag 44 in cpu_relax()

- *Performance regression* on new kernels
  - Only visible on CPU overcommittement, 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

Analyze events for all VMs, all VCPUs:

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

 ©2015 IBM Corporation

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

```
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()
```

```
-static inline void cpu_relax(void)
-{
-	barrier();
-}
+void cpu_relax(void);
```

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

 ©2015 IBM Corporation

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

©2015 IBM Corporation

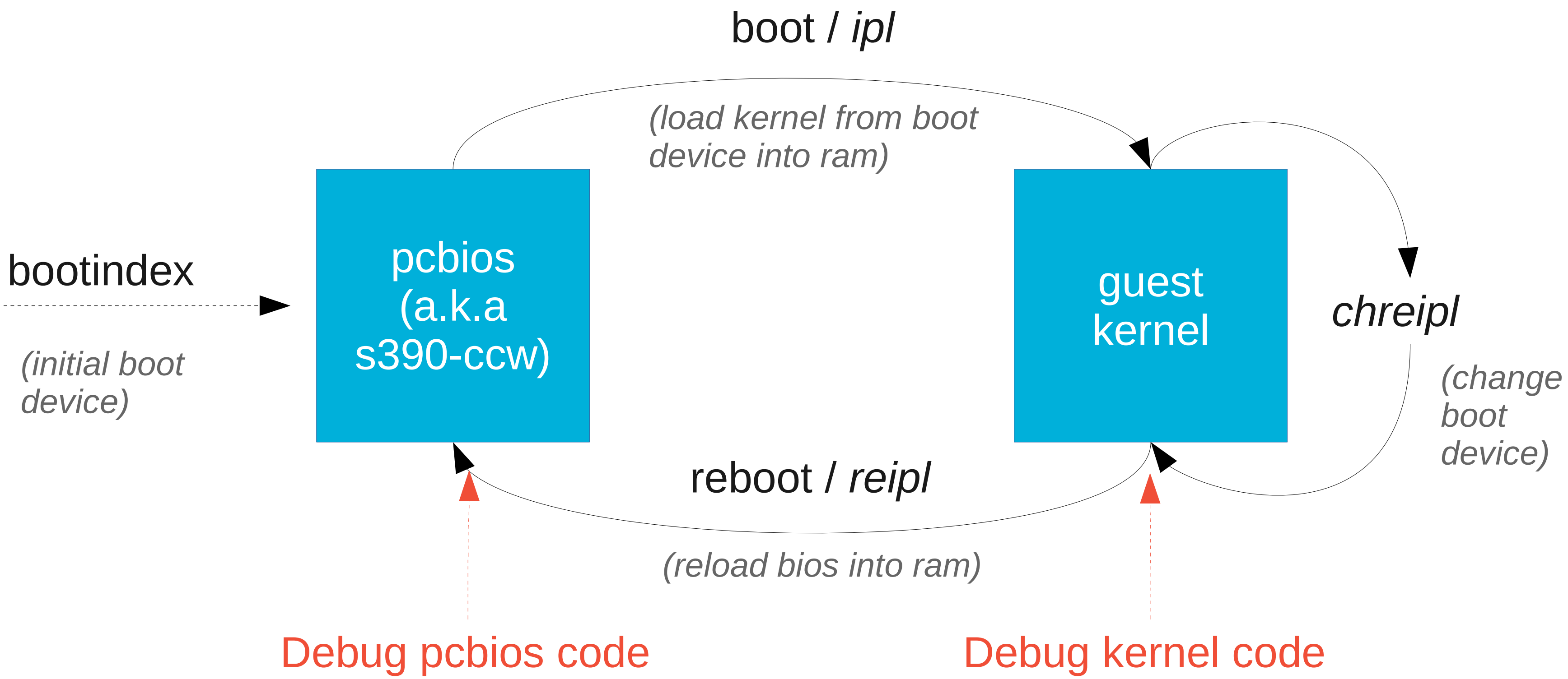
Tips and Tricks (1)

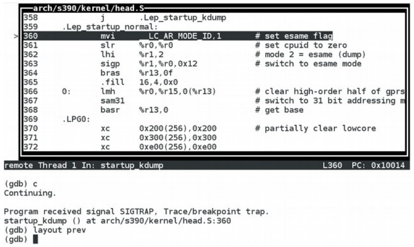
- Ways to *start the QEMU gdbserver*
  - *-s*: Start it directly (can also be passed using libvirt)
  - *-s -S*: Start it, don't start the guest (continue using gdb or QEMU monitor)
  - *Lazily using the QEMU monitor (gdbserver)*
- Access the QEMU monitor using *GDB „monitor“* command

  -> 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

 ©2015 IBM Corporation

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





©2015 IBM Corporation

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)

 ©2015 IBM Corporation

Tips and Tricks (2): just before the transition

```
bootmap.c
   36      static ResetInfo save;
   37
   38      static void jump_to_IPL_2(void)
   39      {
   40          ResetInfo *current = 0;
   41
   42          void (*ipl)(void) = (void *) (uint64_t) current->ipl_continue;
h+ 43          *current = save;
 > 44          ipl(); /* should not return */
   45      }
   46
   47      static void jump_to_IPL_code(uint64_t address)
   48      {
   49          /* store the subsystem information _after_ the bootmap was loaded *
   50          write_subsystem_identification();
remote Thread 1 In: jump_to_IPL_2                              L44   PC: 0x3fe00662
```

```
Program received signal SIGTRAP, Trace/breakpoint trap.
jump_to_IPL_2 () at bootmap.c:42
(gdb) step
(gdb) x current->ipl_continue
0xa050 <.lowcase+38918>:        0x0dd04170
(gdb) step
(gdb) 
```


©2015 IBM Corporation

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) █
```


©2015 IBM Corporation

Thank you!

david.hildenbrand@de.ibm.com

©2015 IBM Corporation

Trademarks

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

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

* 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.

 ©2015 IBM Corporation