Xen Debugging

ORACLE

Zhigang Wang

zhigang.x.wang@oracle.com

November 2009

Agenda

- Debugging requirements
- gdbsx - gdbserver for xen
- kdb for xen
- Xen debug keys
- Guest core dumps
- Xen crash dumps
- Xentrace
- Windows debug
- Xend debug
- Resources
- Q&A

Debugging requirements

- Debugging various xen components:
  - Hypervisor
  - Dom0 kernel
  - Dom0 user space (xend)
  - DomU (pvm/hvm)
- Debugging methods:
  - Dynamic debugging
  - Static debugging (analyze debug core dumps)

gdbsx - gdbserver for xen

- A simple gdbserver stub that runs on dom0 and connects to an unprivileged guest, PV or HVM.
- Makes hypercalls directly without going thru libxc.
- Full standard debugger support via gdb: register/memory read and modifications, single step, etc.
- Support for multiple VCPUs, and single step will execute only one VCPU.

gdbsx - examples

```
# gdbsx -a 5 32 9999    <-- on domain0
$ gdb    <-- on debugging host
(gdb) file vmlinux
(gdb) dir linux-src/
(gdb) target remote <domain0 ip>:9999
(gdb) info thread
[New Thread 1]
  2 Thread 1  0xc04013a7 in hypercall_page ()
  * 1 Thread 0  0xc04013a7 in hypercall_page ()
(gdb) bt
#0  0xc04013a7 in hypercall_page ()
#1  0xc0408664 in raw_safe_halt () at include/asm/mach-
   xen/asm/hypercall.h:197
#2  0xc040321a in xen_idle () at arch/i386/kernel/process-xen.c:109
#3  0xc0403339 in cpu_idle () at arch/i386/kernel/process-xen.c:161
#4  0xc06f49f5 in start_kernel () at init/main.c:618
#5  0xc040006f in startup_32 ()
```

gdbsx - useful gdb macros

```
(gdb) source gdbmacros
(gdb) ps
Pointer     PID    Command
0xc0d5daa0  1      init
0xc0d5d000  2      migration/0
...
(gdb) lsmod
Pointer     Address     Name
0xe1234c00  0xe1231000  xenblk
0xe130b200  0xe12fc000  dm_raid45
0xe120d900  0xe120d000  dm_message
...
(gdb) log
<5>Linux version 2.6.18-128.0.0.0.2.el5xen
(mockbuild@ca-build10.us.oracle.com) (gcc version 4.1.2 20080704 (Red Hat
4.1.2-44)) #1 SMP Wed Jan 21 05:49:36 EST 2009
<6>BIOS-provided physical RAM map:
<4> Xen: 0000000000000000 - 0000000020800000 (usable)
...
```

ORACLE

kdb for xen - features

- Fully implemented in the hypervisor.
- Need serial access to activate and use it.
- earlykdb: to break into kdb early during boot.
- All CPUs are paused, essentially freezing the system.
- Single step will only step that CPU while others stay in kdb.
- Examine domain and vcpu details in pretty print.
- Set breakpoints in the hypervisor.
- Display special registers, irq table, etc.

kdb for xen - entering kdb

- Breaking into kdb: once the serial line is setup, and the system booted with kdb-built hypervisor, ctrl+\ will break into it (requires make kdb=y when compile xen).
- IPI sent to all cpus to pause.
- All cpus enter kdb. One becomes main kdb cpu, while others enter kdb pause mode.
- Switching cpu via the cpu command causes target cpu to become kdb main cpu, while the leaving cpu goes into kdb pause mode.

kdb for xen - commands overview

- **h**: for list of commands
- **dr/mr** : display modify registers
- **dw/dd/mw/md** : display modify words/dwords
- **dwm/ddm**: display words/dwords at machine addr
- **dis** : disassemble instructions
- **f** : display stack frame
- **ni/ss**: single step, over or into functions
- **cpu all** : show cpu status for all cpus
- **dom/vcpu**: display domain/vcpu details
- **gdt/dirq**: display gdt, irq table, etc...

kdb for xen - examples

Type CTRL-\ to break into kdb

```
Enter kdb (cpu:0 reason:1 vcpu=0 domid:32767 eflg:0x246 irqs:1)
ffff828c8017a922: acpi_safe_halt+2          ret
[0]xkdb> cpu all
[0]ffff828c8017a922: acpi_safe_halt+2          ret
[1]ffff828c8017a922: acpi_safe_halt+2          ret
[0]xkdb> dr
(XEN) ----[ Xen-3.4.0  x86_64  debug=n  Not tainted ]----
(XEN) CPU:    0
(XEN) RIP:    e008:[<ffff828c8017a922>] acpi_safe_halt+0x2/0x10
(XEN) RFLAGS: 0000000000000246   CONTEXT: hypervisor
(XEN) rax: 0000000000000003   rbx: 000000008e026c   rcx: 0000000000000001
(XEN) rdx: 0000000000000808   rsi: 0000000004d2b4e8   rdi: ffff83007f2c2460
(XEN) rbp: ffff83007f2c2400   rsp: ffff828c802d7ec0   r8:  00000000000002b9
(XEN) r9:  0000000000000002   r10: 0000000000000000   r11: ffff828c8031a3e0
(XEN) r12: ffff83007f2c2460   r13: 00000b2a9fba595a   r14: 0000000000000000
(XEN) r15: ffff828c8024d100   cr0: 000000008005003b   cr4: 00000000000026f0
(XEN) cr3: 000000020fc0000   cr2: 00000000b7f44000
…
```

kdb for xen - examples continued

```
[0]xkdb> bp schedule
BP set for domid:32767 at: 0xffff828c801178b0 schedule+0
[0]xkdb> go
Breakpoint on cpu 0 at 0xffff828c801178b0
ffff828c801178b0: schedule+0                subq $0x78, %rsp
[0]xkdb> ss
ffff828c801178b4: schedule+4                mov %rbx, 0x48(%rsp)
[0]xkdb> bc all
Deleted breakpoint [0] addr:0xffff828c801178b0 domid:32767
[0]xkdb> cpu 1
Switching to cpu:1
[1]xkdb>
[1]xkdb> f
(XEN) Xen call trace:
(XEN)    [<ffff828c80117a13>] schedule+0x163/0x3a0
(XEN)    [<ffff828c80118968>] do_softirq+0x58/0x80
(XEN)    [<ffff828c8013fedc>] idle_loop+0x4c/0xa0
[1]xkdb> go
```

kdb and gdbsx - status update

- Written and maintained by Mukesh Rathor <mukesh.rathor@oracle.com>
- gdbsx merged to xen-unstable by changeset: 20319:de04fe4e472c on Oct. 15 2009.
- kdb still on http://xenbits.xensource.com/ext/debuggers.hg
- Future work:
  - Hardware watchpoints.
  - Improve timer handling.
  - Some sort of exception handling for very occasional protection faults upon bad data.
- Tests and contributions are welcome.

Xen debug keys - installed handlers

xm debug-key h
xm dmesg

```
...
(XEN) 'h' pressed -> showing installed handlers
(XEN)  key '%' (ascii '25') => Trap to xendbg
(XEN)  key '0' (ascii '30') => dump Dom0 registers
(XEN)  key 'C' (ascii '43') => trigger a crashdump
(XEN)  key 'H' (ascii '48') => dump heap info
(XEN)  key 'N' (ascii '4e') => NMI statistics
(XEN)  key 'Q' (ascii '51') => dump PCI devices
(XEN)  key 'R' (ascii '52') => reboot machine
(XEN)  key 'a' (ascii '61') => dump timer queues
(XEN)  key 'c' (ascii '63') => dump cx structures
(XEN)  key 'd' (ascii '64') => dump registers
(XEN)  key 'e' (ascii '65') => dump evtchn info
(XEN)  key 'h' (ascii '68') => show this message
(XEN)  key 'i' (ascii '69') => dump interrupt bindings
(XEN)  key 'm' (ascii '6d') => memory info
…
```

Xen debug keys - examples

```
# xm debug-key d
# xm dmesg
...
(XEN) 'd' pressed -> dumping registers
(XEN)
(XEN) *** Dumping CPU1 host state: ***
(XEN) ----[ Xen-3.4.0  x86_64  debug=n  Not tainted ]----
(XEN) CPU:    1
(XEN) RIP:    e008:[<ffff828c8010d291>] __dump_execstate+0x1/0x60
(XEN) RFLAGS: 0000000000210246   CONTEXT: hypervisor
(XEN) rax: 0000000000000000   rbx: 0000000000000064   rcx: 0000000000200046
(XEN) rdx: 000000000000000a   rsi: 000000000000000a   rdi: 0000000000000000
(XEN) rbp: ffff83007f0f7f28   rsp: ffff83007f0f7d98   r8:  0000000000000001
XEN) r9:  0000000000000001   r10: 00000000fffffffc   r11: ffff828c8012d020
(XEN) r12: ffff828c802e48a0   r13: ffff83007f0f7f28   r14: 0000000000000000
(XEN) r15: 0000000000000000   cr0: 000000008005003b   cr4: 00000000000026f0
…
```

Guest core dumps

- Get manually: xm dump-core
- Xen support automatic core dump. On vm.cfg:
  - on_crash = ‘coredump_restart’
  - on_crash = ‘coredump_destroy’
- Guest crash dump generated cores.
- Dom0 and hypervisor crash dump generated cores.

Xen crash dumps

- Using kexec and kdump ported from Linux.
- Provide a crash-dump facility for both domain 0 and the hypervisor.
- Kexec can reboot into xen or into a Linux kernel.
- Kdump under xen is similar to the standard Linux implementation.
  - Both dom0 kernel panic and hypervisor panic are supported.
  - dom0 loads the secondary “crash kernel”.
  - The secondary kernel starts if dom0 panics or xen panics.
  - The physical memory range is reserved in the hypervisor.
  - Range is reserved using xen command line options: crashkernel=Y@X

Guest core dumps analyze

```
$ crash vmlinux core-OVM_EL5U3_X86_PVM_4GB
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM     VSZ    RSS  COMM
>      0      0   0  c06762c0  RU   0.0       0      0  [swapper]
>      0      1   1  c0d5d550  RU   0.0       0      0  [swapper]
       1      0   0  c0d5daa0  IN   0.1    2080    604  init
       2      1   0  c0d5d000  IN   0.0       0      0  [migration/0]
       3      1   0  c0d54aa0  IN   0.0       0      0  [ksoftirqd/0]
...
crash> mod
 MODULE    NAME              SIZE  OBJECT FILE
e1206b80  scsi_dh          11713  (not loaded)  [CONFIG_KALLSYMS]
e120a500  dm_mem_cache     10049  (not loaded)  [CONFIG_KALLSYMS]
e120d900  dm_message        6977  (not loaded)  [CONFIG_KALLSYMS]
...
```

Xen crash dumps analyze

```
$ crash xen-syms-2.6.18-128.el5 xen-syms-2.6.18-128.el5.debug core-EL5U3
crash> doms
   DID   DOMAIN  ST T  MAXPAGE  TOTPAGE VCPU SHARED_I  P2M_MFN
  32753 ff21c080 RU O     0        0      0      0      ----
  32754 ff1c8080 RU X     0        0      0      0      ----
  32767 ff214080 RU I     0        0      2      0      ----
>*    0 ffbf0080 RU 0 ffffffff   67100    2  ffbec000   3e308
crash> pcpus
   PCID   PCPU   CUR-VCPU    TSS
 *    0 ff1d3fb4 ffbe7080 ff1fa180
      1 ff21bfb4 ffbe6080 ff1fa200
crash> vcpus
   VCID  PCID   VCPU   ST T DOMID  DOMAIN
      0     0 ffbfd080 RU I 32767 ff214080
      1     1 ff1cc080 RU I 32767 ff214080
>*    0     0 ffbe7080 RU 0     0 ffbf0080
>     1     1 ffbe6080 RU 0     0 ffbf0080
```

Xentrace - capture trace buffer data

- Get trace data:

  # **xentrace -D -S 256 -T 1 trace.raw**

- Analyze using xentrace_format:

  # **cat trace.raw | xentrace_format formats >trace.txt**

```
…
CPU0  16939048793256 (+       0)  do_block          [ domid = 0x00000000, edomid = 0x00000000 ]
CPU0  16939048795152 (+    1896)  switch_infprev    [ old_domid = 0x00000000, runtime = 610046 ]
CPU0  16939048795536 (+     384)  switch_infnext    [ new_domid = 0x00007fff, time = 610046, r_time = 4294967295 ]
CPU0  16939048795912 (+     376)  running_to_blocked  [ dom:vcpu = 0x00000000 ]
CPU0  16939048796272 (+     360)  runnable_to_running [ dom:vcpu = 0x7fff0000 ]
CPU0  16939048797048 (+     776)  __enter_scheduler [ prev<domid:edomid> = 0x00000000 : 0x00000000, next<domid:edomid> = 0x00007fff : 0x00000000 ]
…
```

Xentrace - analyze using xenalyze

$ **xenalyze --cpu-hz=1.8G --dump-all /tmp/test.trace | less -B -b 4096**

```
...
]  1.000744383 -x  vmexit exit_reason EXCEPTION_NMI eip fffff80001029795
]  1.000744383 -x mmio_assist w gpa fee000b0 data 0
]  1.000745561 x-  vmexit exit_reason EXCEPTION_NMI eip fffff8000102ff73
]  1.000745561 x- mmio_assist w gpa fee000b0 data 0  vlapic eoi
]  1.000745561 x- fast mmio va fffffffffffe00b0
]  1.000748415 -x  vmentry
]  1.000748692 x-  vmentry
]  1.000749948 -x  vmexit exit_reason HLT eip fffffadffab2fb41
]  1.000749948 -x  hlt [ 0 0 a0028006 10dced2d ]
]  1.000750155 x-  vmexit exit_reason HLT eip fffffadffab2fb41
]  1.000750155 x-  hlt [ 0 0 a0028006 10dcef1c ]
…
```

Xenalyze by George Dunlap: http://xenbits.xensource.com/ext/xenalyze.hg

Xentrace - guest context

```
# xenctx --symbol-table=System.map-2.6.18-128.0.0.0.2.el5xen --all 1
cs:eip: 0061:c04013a7 hypercall_page+0x3a7
flags: 00001246 i z p
ss:esp: 0069:c06effc0
eax: 00000000   ebx: 00000001   ecx: 00000000   edx: 00000000
...
Stack:
 c0408664 c141c6c8 ffffffff c040321a c0403339 c141c6c8 c071fc64 c0627831
 c06f49f5 00004b64 c0765800 01020800 c0dcb000 00000000 00000000 c040006f

Call Trace:
  [<c04013a7>] hypercall_page+0x3a7  <--
  [<c0408664>] raw_safe_halt+0x8c
  [<c040321a>] xen_idle+0x22
  [<c0403339>] cpu_idle+0x91
  [<c06f49f5>] start_kernel+0x37a
```

Xentrace - hvm context

```
# xen-hvmctx 2
HVM save record for domain 2
Entry 0: type 1 instance 0, length 24
    Header: magic 0x54381286, version 1
          Xen changeset ffffffffffffffff
          CPUID[0][%eax] 0x000006f2
Entry 1: type 2 instance 0, length 1016
   CPU:    rax 0x0000000000000000     rbx 0x0000000000000000
         rcx 0x00000000c0403bb0     rdx 0x00000000c06f0000
         rbp 0x00000000c0627743     rsi 0x00000000c072dee4
         rdi 0x00000000c0627765     rsp 0x00000000c06f0fd8
          r8 0x0000000000000000      r9 0x0000000000000000
         r10 0x0000000000000000     r11 0x0000000000000000
         r12 0x0000000000000000     r13 0x0000000000000000
         r14 0x0000000000000000     r15 0x0000000000000000
         rip 0x00000000c0403be1  rflags 0x0000000000000246
…
```

Windows debug

- Dynamic debug with WinDbg
  - Enable debug mode on Windows guest.
  - In vm.cfg: serial = 'tcp:<Debug Host IP>:<Port>'
  - On debug host: start HW VSP and transfer IP package to virtual COM port. Then start WinDbg to open virtual COM port to control.
- Analyze windows core dumps
  - Tool to convert ELF core dump to WinDbg recognized Windows kernel memory dump.

HW VSP: http://www.hw-group.com/products/hw_vsp/index_en.html

Xend debug

- Xend is written in **Python** and can be debugged by **pdb**.
- All user space debugging techniques are applied.
- Xend support tracing.
- Refer to: http://wiki.xensource.com/xenwiki/XenDebugging.

Conclusion

- Xen has full-featured debug support for every component.
- Leverage existing techniques as much as possible.
- Documentations need improvement.

Resources

- Xen project: http://xen.org/
- Xen mailing list: http://lists.xensource.com/
- Xen Wiki: http://wiki.xensource.com/xenwiki/
- Xen unstable:: http://xenbits.xensource.com/xen-unstable.hg
- Xen Debuggers: http://xenbits.xensource.com/ext/debuggers.hg
- Xen Serial Console: http://os-drive.com/wiki/XenSerialConsole
- Xen Debugging: http://os-drive.com/wiki/XenDebugging
- Debugging Xen and Xen guests: http://www.xen.org/files/xensummitboston08/Mukesh%20xendbg-present.pdf
- Xen port of kexec/kdump: http://www.xen.org/files/summit_3/kexec_kdump.pdf
- Crash: http://people.redhat.com/anderson/
- GDB: http://www.gnu.org/software/gdb/
- Oracle VM: http://oracle.com/virtualization
- Oracle Enterprise Linux: http://www.oracle.com/linux

Q U E S T I O N S
A N S W E R S