The State of Kernel Debugging Technology

Jason Wessel
- Product Architect for WR Linux Core Runtime
- Kernel.org KDB/KGDB Maintainer

August 12th, 2010

WIND RIVER

Agenda

- Brief history of kernel.org kernel debuggers
- "crash" course in KDB
- Ideas for the future of the kernel debugger

*** Presentation/code found at: http://kgdb.wiki.kernel.org ***

Is there anything better than KGDB?

- Good
  - KGDB / KDB
- Better
  - QEMU/KVM OR Virtual box OR vmware backend debugger
  - kdump/kexec
- Best
  - ICE / JTAG (usb or ethernet)
  - Simics - www.simics.com (because it has backward stepping)
- In a class by itself
  - printk() / trace_printk()

The challenge is knowing what to use when...

Brief History of kernel debugger

- 2008-2009
  - 2.6.26 – KGDB “light” merged (just x86 and ARM)
  - 2.6.27 – MIPS and PowerPC
  - Added KGDB support for sparc, blackfin and sh
- 2010
  - 2.6.35
    - KDB merged to mainline
    - Early debug with EHCI debug port or keyboard + vga console
  - 2.6.36
    - microblaze arch support
    - ftrace dump support via KDB/KGDB
    - Atomic KMS (Kernel Mode Setting) API merged

EHCI Debug Port

- Great for when you do not have rs232
- Higher speed than rs232
- Works with KGDB

  kgdbdbgp=0
- Use it as a Linux Console

  console=ttyUSB0 AND/OR earlyprintk=kdbgp0

- Read more in your kernel source tree:

  Documentation/x86/earlyprintk.txt
- You can buy one at

  http://www.semiconductorstore.com/cart/pc/viewPrd.asp?idproduct=12083

KDB – kernel debug shell History

- The goal of the merge KDB and KGDB was simple:
    - Unify the fragmented kernel debugger communities
- KDB was a derived from from the 10 year old project:
    - ftp://oss.sgi.com/projects/kdb/download/v4.4/
- The merge work started in 2009 with many prototypes
    - Originally KDB was > 64,000 lines of changes for just x86
    - After some significant gutting of anything that was common, the result was a platform independent KDB hooked up to the same infrastructure (debug_core) that is used by KGDB.
    - The final KDB patch set was < 8500 lines of changes
- For more information about differences in SGI KDB vs mainline KDB
    - https://kgdb.wiki.kernel.org/index.php/KDB_FAQ

WIND RIVER

KDB – The in-kernel debug shell

- To use KDB you must meet one of following constraints
  - Use a non usb keyboard + vga text console
  - Use a serial port console
  - Use a USB EHCI debug port and debug dongle
- KDB is not a source debugger
  - However you can use it in conjunction with gdb and an external symbol file
- Maybe you don't need a kernel debugger, but you at least want a chance to see ftrace logs, dmesg, poke a stack trace or do one final sysrq.
  - KDB might still be the tool you are looking for

WIND RIVER

Loading KDB

Having KDB loaded allows you to trap the panic handler.

- For a serial port:

```
echo ttyS0 > /sys/module/kgdboc/kernel/kgdboc
```

- For the keyboard + vga text console

```
echo kbd > /sys/module/kgdboc/kernel/kgdboc
```

- Enter KDB with sysrq-g

```
echo g > /proc/sysrq-trigger
```

- Remember KDB is a stop mode debugger
  - Entering KDB means all the other processors skid to a stop
  - You can run some things like: lsmod, ps, kill, dmesg, bt
  - ftdump to dump ftrace logs (not merged to mainline yet)
  - You can also use hw breakpoints or modify memory

WIND RIVER

KDB “crash” course

- Simply loading KDB gives you the opportunity to stop and look at faults perhaps using external tools

```
echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
insmod test_panic.ko
echo 1 > /proc/test_panic/panic
```

- After the panic collect dmesg, ftdump, bt, and lsmod
- Use gdb to load the symbol file and kernel module

```
gdb ./vmlinux
add-symbol-file test_panic.ko ADDR_FROM_LSMOD
info line *0xADDR_FROM_BT
```

WIND RIVER

Pre-recorded Demonstration 1

- Example of a useless call to panic()
    - http://www.youtube.com/watch?v=V6Qc8ppJ_jc
- Example of finding the useless call to panic()
    - http://www.youtube.com/watch?v=LqAhY8K3XzI

KDB Demonstration 2 - breakpoints

- Load KDB and use a data write breakpoint

```
insmod test_panic.ko
echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
echo g > /proc/sysrq-trigger
bph tp_address_ref dataw
go
```

- Cause the problem and collect the data

```
echo 1 > /proc/test_panic/bad_access
bt
rd
lsmod
```

- Statically look at the source with gdb + module address

Pre-recorded Demonstration 2

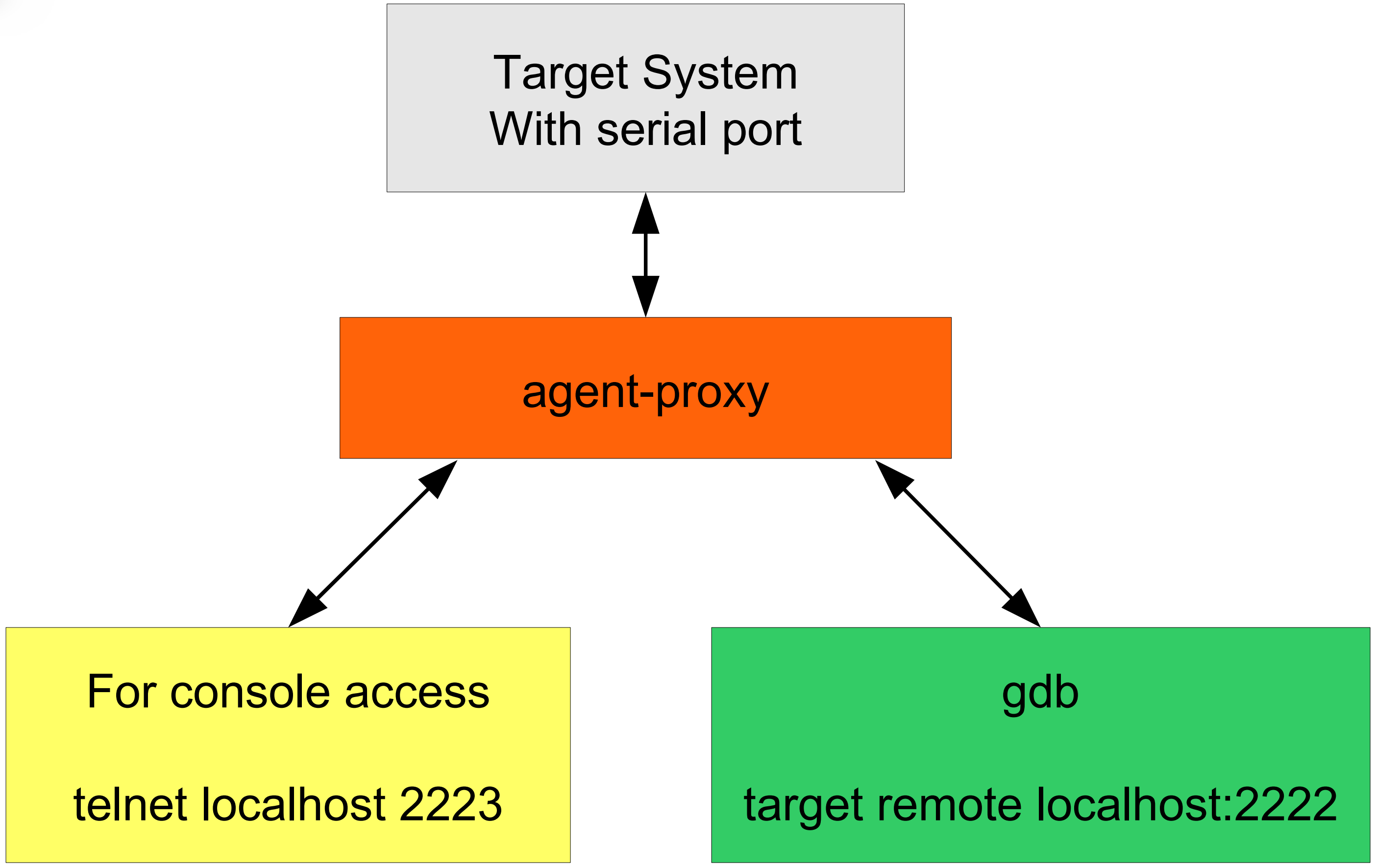
- Example of a kernel bad paging request
  - http://www.youtube.com/watch?v=bBEh_UduX04
- Example of using HW breakpoint in kdb
  - http://www.youtube.com/watch?v=MfJU2E0aJwg

WIND RIVER

Remember KDB is KGDB too!

- If you only have a single serial port, it just got easier to use KGDB if you want to use it.
- Try the agent-proxy
- The agent-proxy is nothing more then a tty → tcp connection mux that can allow you to connect more than one client application to a tty
- You can even use the agent-proxy with the EHCI debug port device.

Sharing the console - kgdboc



KGDB demonstration setup

- Use a connection multiplexer
  - By default you can only connect one application at a time to the console
  - In the case of kgdboc you want an interactive console & a debug port

agent-proxy *CONSOLE_PORT*^DEBUG_PORT IP_ADDR PORT

- More or less turns your local serial port into a terminal server

```
agent-proxy 2223^2222 0 /dev/ttyS0,115200
```

- Use it to multiplex a remote terminal server or simulator connection

```
agent-proxy 2223^2222 128.224.50.38 8181
```

- The agent-proxy is now available:

```
git clone git://git.kernel.org/pub/scm/utils/kernel/kgdb/agent-proxy.git
cd agent-proxy ; make
```

KGDB demonstration

- On the target system

```
echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc
insmod test_panic.ko
```

- In gdb

```
tar remote localhost:2222
break sys_sync
c
```

- On the target

```
sync
```

- In gdb

```
awatch tp_address_ref
inf br
c
```

- On the target

```
echo 1 > /proc/test_panic/bad_access
```

- Back to gdb where we can pass along the exception
- signal 9

WIND RIVER

Pre-recorded Demonstration 3

- Start up the agent-proxy and connect and hit a breakpoint a sys_sync
  - http://www.youtube.com/watch?v=sWiHV5mt8_k
- Data Access breakpoint on tp_address_ref
  - http://www.youtube.com/watch?v=nnopzcwvLTs

WIND RIVER

Future plans

- More drivers and bug fixes for atomic kernel mode setting
- Continue to improve the non ehci debug usb console
- Improve keyboard panic handler
- Further integration with kprobes and hw assisted debugging
- netconsole / kgdboe v2 – Use dedicated HW queues
- ...wild, far off ideas...
    - source stepping in KDB
    - user space backtrace
    - Individual thread and cpu run control

WIND RIVER

References

- KGDB/KDB Website

  http://kgdb.wiki.kernel.org

- KGDB/KDB Mailing list
  - kgdb-bugreport@lists.sourceforge.net
  - https://lists.sourceforge.net/lists/listinfo/kgdb-bugreport
- Source code used in this presentation
  - The 2.6.36 kernel was used
  - The kernel module code can be found at:

  http://kernel.org/pub/linux/kernel/people/jwessel/dbg_webinar/crash_mod.tar.bz2

KGDB facts

- KGDB and KDB use the same debug backend
- kgdboe (KGDB over ethernet) is not always reliable
    - kgdboe in the current form WILL NOT BE MAINLINED
    - Linux IRQs can get preempted and hold locks making it unsafe or impossible for the polled ethernet driver to run
    - Some ethernet drivers are so complex with separate kernel thread that the polled mode ethernet can hang due to locking or unsafe HW resource access
    - If you really want to attempt use kgdboe successfully, use a dedicated interface if you have one and do not use kernel soft or hard IRQ preemption.
- kgdboc is slow but the most reliable
- The EHCI debug port is currently the fastest KGDB connection

WIND RIVER

WIND RIVER