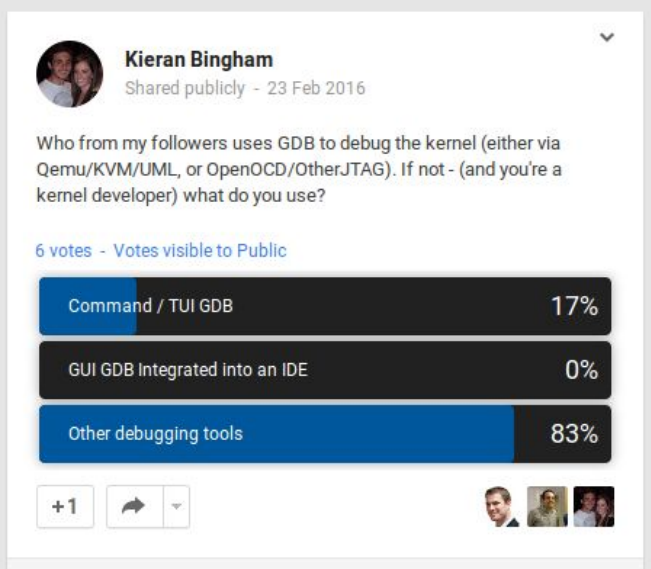
Debugging the Linux Kernel with GDB

Kieran Bingham

Debugging the Linux Kernel with GDB

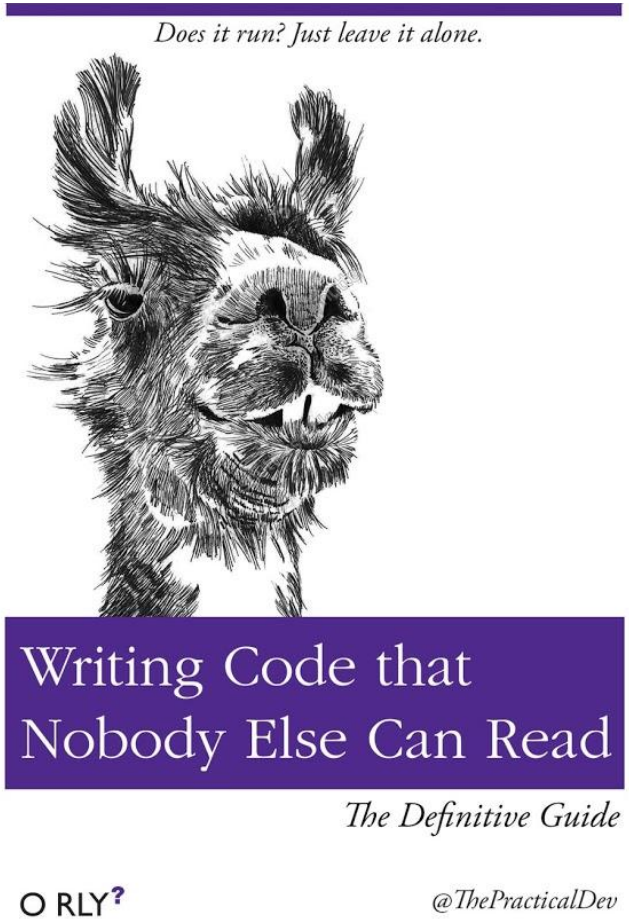
- Many of us need to debug the Linux kernel
- Proprietary tools like Trace32 and DS-5 are $$$
- Open source debuggers like GDB lack ‘kernel awareness’ features found in proprietary tools

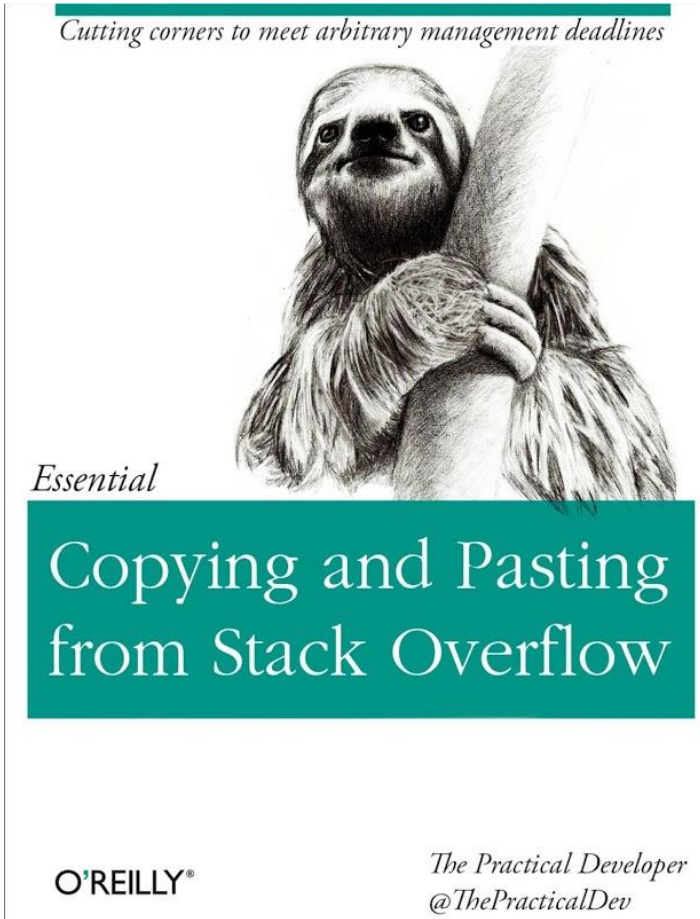
- What exists today
- How you can use it to get data
- How can we make it better



Linaro

{They, we} wouldn't … would {they, we} ?





Linaro

Linus (~2000)

I don't like debuggers. Never have, probably never will. I use gdb all the time, but I tend to use it not as a debugger, but as a disassembler on steroids that you can program.

You can use a kernel debugger if you want to, and I won't give you the cold shoulder because you have "sullied" yourself. But I'm not going to help you use one, and I would frankly prefer people not to use kernel debuggers that much.

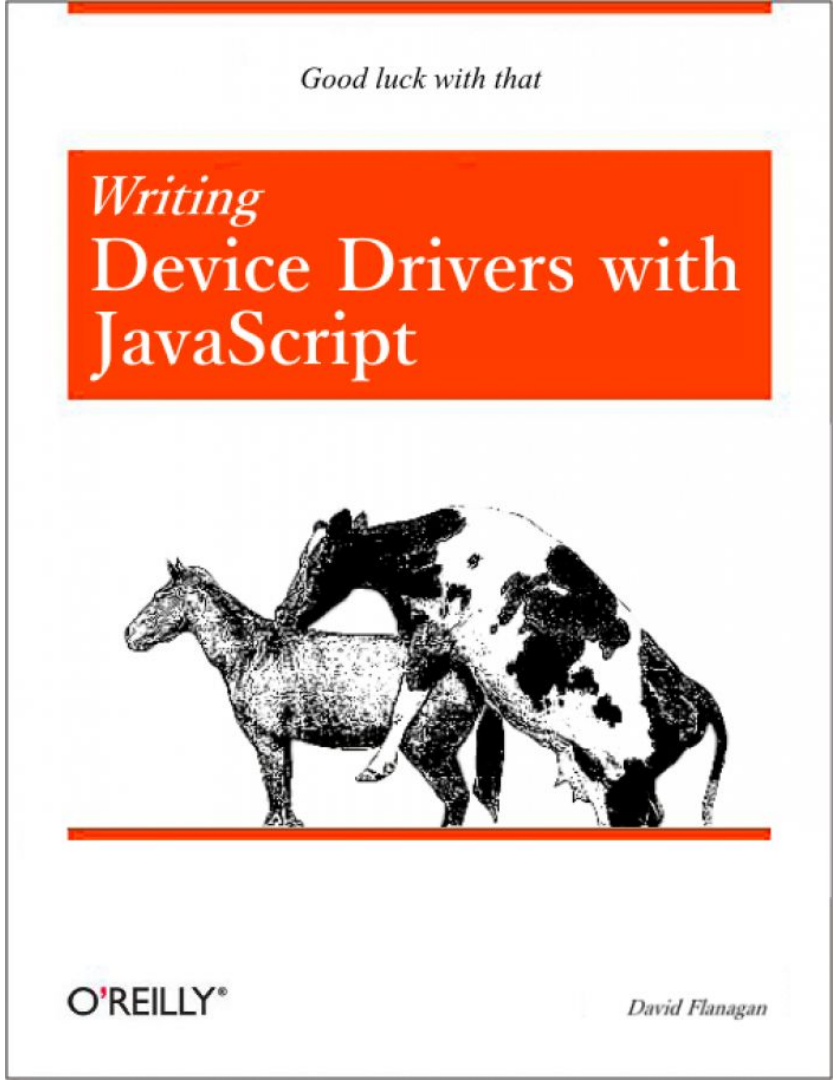
http://lwn.net/2000/0914/a/lt-debugger.php3

Why?

There is always code to debug.

We don’t always write it ourselves.

But we do have to fix it



Linaro

How can we improve free tools

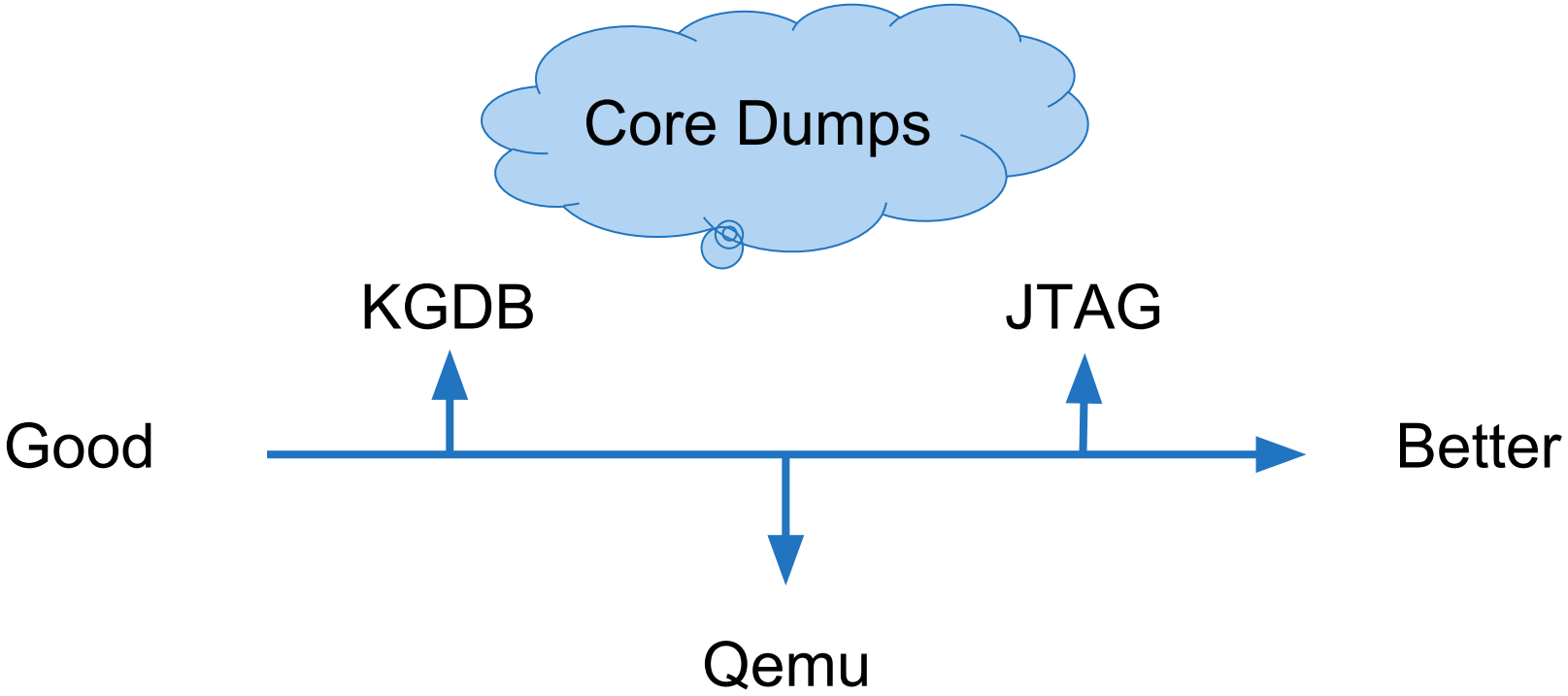
- Both LLDB and GDB have extension capabilities
- Jan Kizka has led the way, adding Kernel support for GDB
- OpenOCD provides free JTAG connectivity
- Automated testing needed

Coming Up

- Target options
  - KGDB
  - QEmu/KVM/UML
  - JTAG
  - Core Dumps
- Linux Awareness
  - Thread Awareness
  - Module Support
  - Data retrieval
  - Extending with Python
- Q+A

Targets for debugging Linux with GDB

- GDB client using the gdbremote protocol
  a. Connection to a KGDB stub in a running kernel
  b. Connect to a QEmu stub running a virtual kernel environment
  c. To a gdbremote compliant JTAG probe, such as OpenOCD
- GDB session on host
  a. Core Dump file
  b. UML Kernel

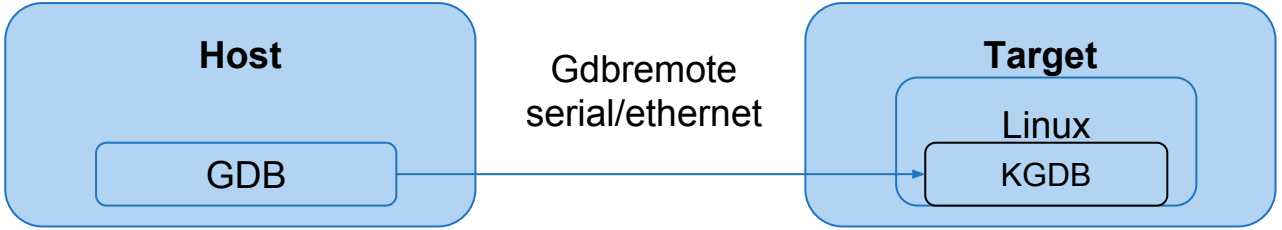


Targets: KGDB with GDB

- Debug stub in the kernel compliant with gdbremote protocol
  - Enable with CONFIG_KGDB

+ Already supported on many platforms
+ **All kernel threads enumerated in GDB (via gdbremote)**

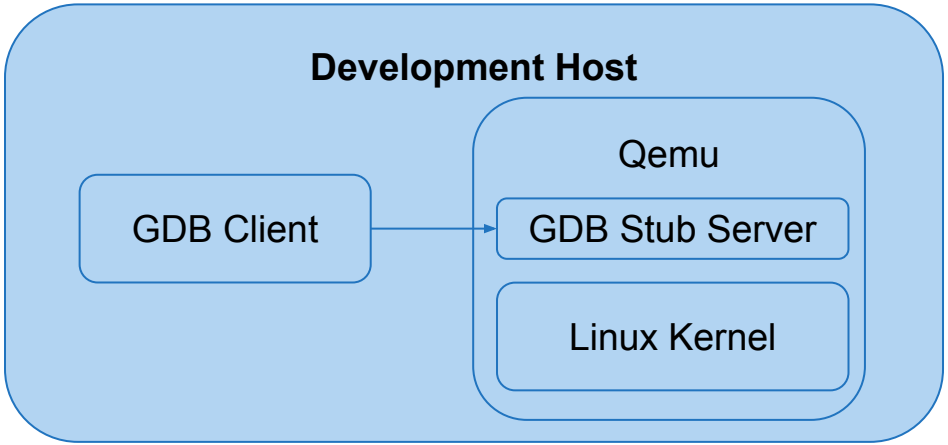
- Requires cooperation between debugger and kernel stub
  - Less suitable for serious crashes
- Isn’t enabled on production systems
- Requires enough support for serial or ethernet



Targets: QEmu

+ Qemu is open source and has gdbremote stub
+ No ‘real’ hardware required
+ Good for testing generic kernel code on many architectures
+ **Good environment for developing Kernel Awareness extensions**

- Unlikely to be useful for SoC or board related issues



Targets : Qemu (Example)

```
qemu-system-arm -kernel ./zImage -dtb ./vexpress-v2p-ca15-tc1.dtb -M vexpress-a15 -smp 2 -m 1024 -append 'root=/dev/nfs nfsroot=10.0.2.2:/opt/root/armv7/,tcp,v3 rw
ip=dhcp mem=1024M raid=noautodetect rootwait console=ttyAMA0,38400n8 devtmpfs.mount=0'     -nographic -gdb tcp::32770
[          0.000000] Booting Linux on physical CPU 0x0
[          0.000000] Linux version 4.6.0-rc1 (kbingham@CookieMonster) (gcc version 5.2.1 20151010 (Ubuntu 5.2.1-22ubuntu1) ) #13 SMP Thu Mar 31 10:33:19 BST 2016
[          0.000000] CPU: ARMv7 Processor [412fc0f1] revision 1 (ARMv7), cr=10c5387d
[          0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[          0.000000] Machine model: V2P-CA15
[ ..... ]
[          3.989042] IP-Config: Got DHCP answer from 10.0.2.2, my address is 10.0.2.15
[          3.991451] IP-Config: Complete:
[          3.991672]          device=eth0, hwaddr=52:54:00:12:34:56, ipaddr=10.0.2.15, mask=255.255.255.0, gw=10.0.2.2
[          3.991900]          host=10.0.2.15, domain=, nis-domain=(none)
[          3.992039]          bootserver=10.0.2.2, rootserver=10.0.2.2, rootpath=    nameserver0=10.0.2.3
```

```
arm-linux-gdb ./linux/vmlinux  -iex 'add-auto-load-safe-path ./linux' -ex 'target remote localhost:32770'
Remote debugging using localhost:32770
cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
74          ret         lr
(gdb) info threads
  Id   Target Id       Frame
* 1          Thread 1 (CPU#0 [halted ]) cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
  2          Thread 2 (CPU#1 [halted ]) cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
(gdb) bt
#0  cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
#1  0xc0308728 in arch_cpu_idle () at /home/lkd/sources/linux/arch/arm/kernel/process.c:72
#2  0xc0376b28 in cpuidle_idle_call () at /home/lkd/sources/linux/kernel/sched/idle.c:151
#3  cpu_idle_loop () at /home/lkd/sources/linux/kernel/sched/idle.c:242
#4  cpu_startup_entry (state=<optimized out>) at /home/lkd/sources/linux/kernel/sched/idle.c:291
#5  0xc0ae8a30 in rest_init () at /home/lkd/sources/linux/init/main.c:408
#6  0xc0f00c5c in start_kernel () at /home/lkd/sources/linux/init/main.c:661
```

Targets : Qemu (Example)

```
qemu-system-arm -kernel ./zImage -dtb ./vexpress-v2p-ca15-tc1.dtb -M vexpress-a15 -smp 2 -m 1024 -append 'root=/dev/nfs nfsroot=10.0.2.2:/opt/root/armv7/,tcp,v3 rw
ip=dhcp mem=1024M raid=noautodetect rootwait console=ttyAMA0,38400n8 devtmpfs.mount=0'     -nographic -gdb tcp::32770
[            0.000000] Booting Linux on physical CPU 0x0
[            0.000000] Linux version 4.6.0-rc1 (kbingham@CookieMonster) (gcc version 5.2.1 20151010 (Ubuntu 5.2.1-22ubuntu1) ) #13 SMP Thu Mar 31 10:33:19 BST 2016
[            0.000000] CPU: ARMv7 Processor [412fc0f1] revision 1 (ARMv7), cr=10c5387d
[            0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[
[ ..... ]
[
[
[
[
[
```

QEmu is a user process trying to mount NFS on ports above 1024
This isn't allowed by default, so we need to add the *'insecure'* option

```
$ cat /etc/exports
/opt/rootfs                    *(rw,sync,no_subtree_check,no_root_squash,insecure)
```

```
arm-l
Remo
cpu_v
74
(gdb)
  Id   T
* 1
  2        Thread 2 (CPU#1 [halted ]) cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
(gdb) bt
#0  cpu_v7_do_idle () at /home/lkd/sources/linux/arch/arm/mm/proc-v7.S:74
#1  0xc0308728 in arch_cpu_idle () at /home/lkd/sources/linux/arch/arm/kernel/process.c:72
#2  0xc0376b28 in cpuidle_idle_call () at /home/lkd/sources/linux/kernel/sched/idle.c:151
#3  cpu_idle_loop () at /home/lkd/sources/linux/kernel/sched/idle.c:242
#4  cpu_startup_entry (state=<optimized out>) at /home/lkd/sources/linux/kernel/sched/idle.c:291
#5  0xc0ae8a30 in rest_init () at /home/lkd/sources/linux/init/main.c:408
#6  0xc0f00c5c in start_kernel () at /home/lkd/sources/linux/init/main.c:661
```

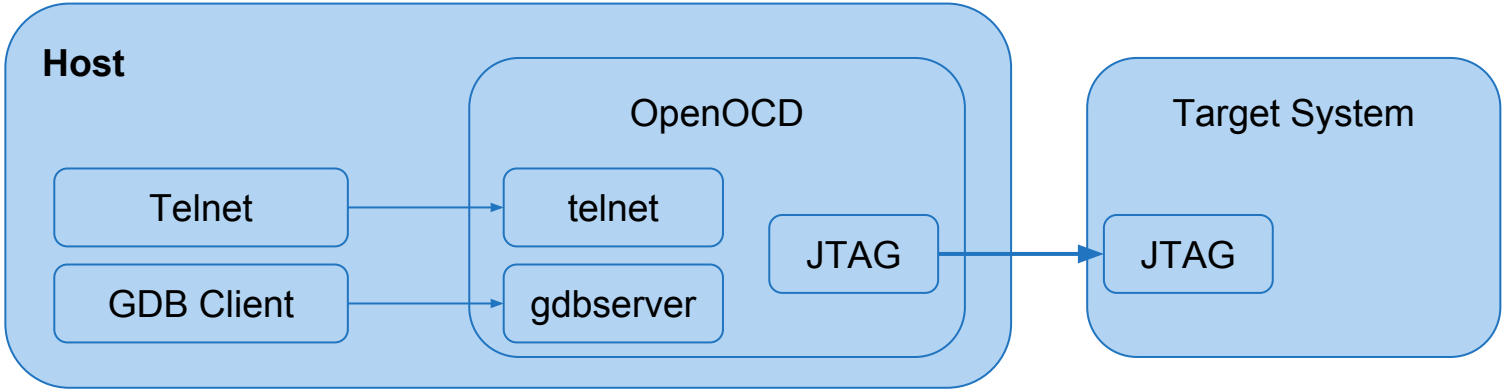
Linaro

Targets: JTAG

- + OpenOCD is open source
- + Supports gdbremote protocol
- + Supports many ARM/MIPS CPUs
- + Supports many FTDI based JTAG probes

http://openocd.org

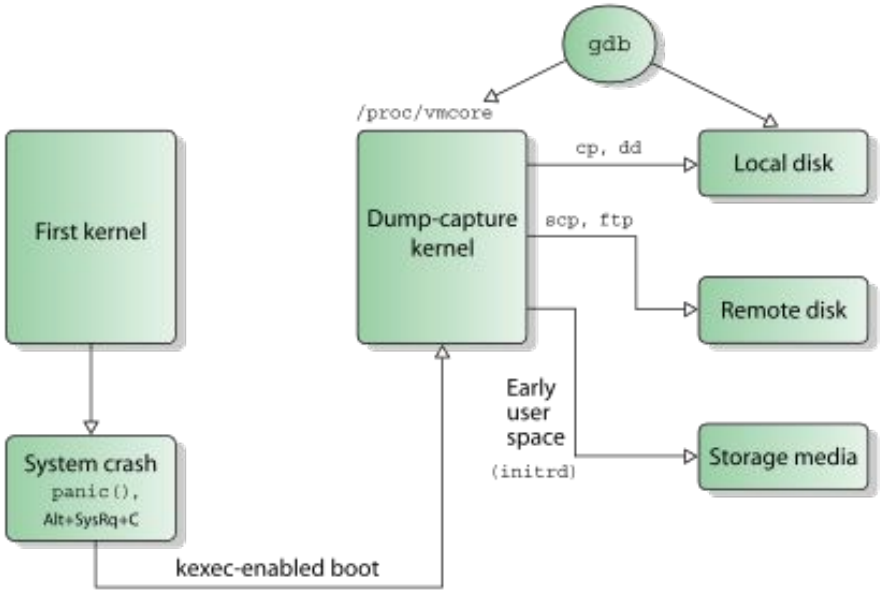
http://elinux.org/JTAG



Targets: Core Dumps

- CONFIG_PROC_KCORE
  - sudo gdb vmlinux /proc/kcore
  - Virtual ELF core file of live kernel
  - No modifications can be made

- CONFIG_PROC_VMCORE
  - /proc/vmcore
  - Used in conjunction with kexec, kdump and the crash utility from RedHat
  - py-crash, and libkdumpfile support coming to GDB from SUSE



https://en.wikipedia.org/wiki/Kdump_(Linux) @V4711

https://www.suse.com/documentation/sles-12/book_sle_tuning/data/part_tuning_dumps.html

Linux Awareness

- Provide the debugger with additional knowledge of the underlying operating system to enable a better debugging experience.
  - Where is the Task List?
  - What is in the Kernel Log Buffer?
  - What modules are loaded? Where?

- We split Linux Awareness into three areas
  1. Task Awareness
     - Ability to report all task_structs as threads in GDB
     - Provides selectable GDB threads with context commands
  2. Loadable Module Support
     - Hooks for automatic symbol resolution when modules are inserted
  3. OS Helper Commands
     - Interacting with the debugger to obtain useful information

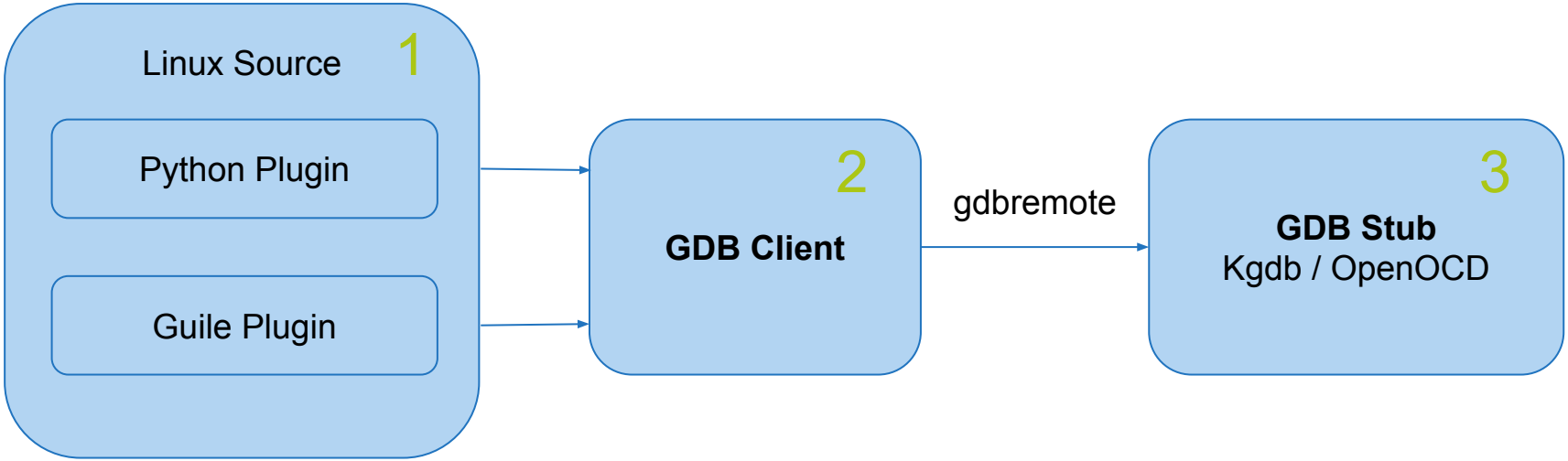
GDB C Extension - Linux Kernel Debugger (LKD)

- Original tools written at ST Micro provide “Linux Awareness”
- ST-LKD based on GDB 7.6
- Developed for STMC2 JTAG debugger

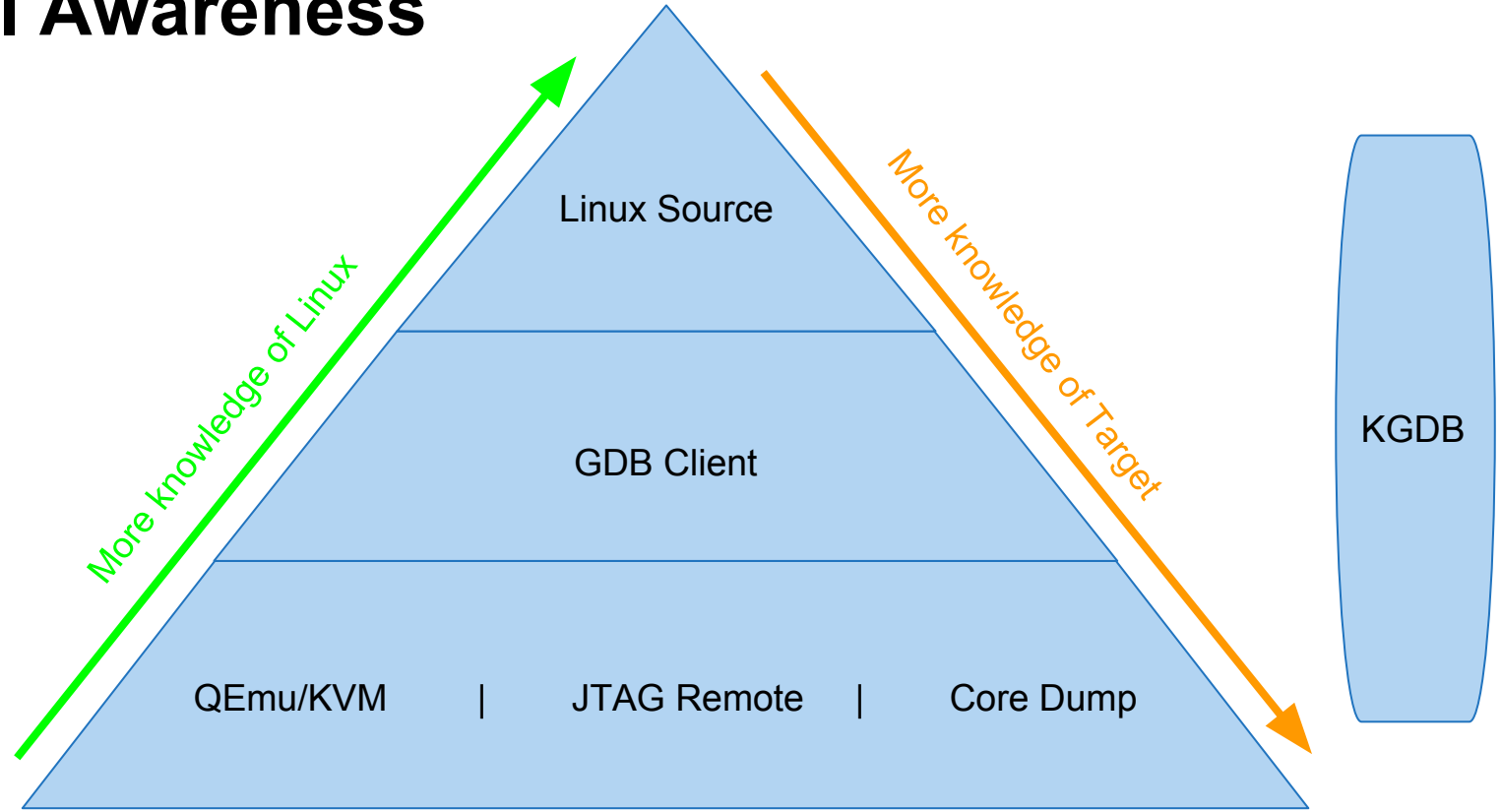
- Upstream project started by Peter Griffin, supported by ST and Linaro

Where to put the ‘awareness’

1. Scripting in GDB (Python/Guile)
2. C extension in GDB
3. Awareness in GDB Stub



Kernel Awareness



LKD-C vs LKD-Python

LKD-C

+ Reference code available
+ Working now

- Puts Linux specific code into GDB

LKD-Python

+ Awareness lives in source tree
+ Generic approach for other OS's
+ Or languages ….

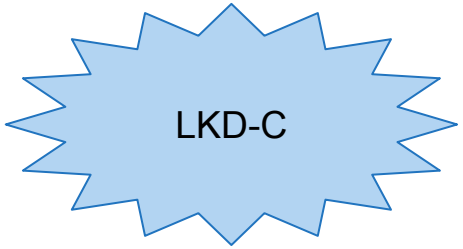
- gdb.Target layer exposes gdb internal hooks to the outside
- Must be robust!

Thread Awareness : GDB Target Implementation

```
static struct target_ops * linux_kthread_target (void)
{
  struct target_ops *t = XCNEW (struct target_ops);

  t->to_shortname = "linux-kthreads";
  t->to_longname = "linux kernel-level threads";
  t->to_doc = "Linux kernel-level threads";
  t->to_close = linux_kthread_close;
  t->to_mourn_inferior = linux_kthread_mourn_inferior;
  t->to_fetch_registers = linux_kthread_fetch_registers;
  t->to_store_registers = linux_kthread_store_registers;
  t->to_wait = linux_kthread_wait;
  t->to_resume = linux_kthread_resume;
  t->to_thread_alive = linux_kthread_thread_alive;
  t->to_update_thread_list = linux_kthread_update_thread_list;
  t->to_extra_thread_info = linux_kthread_extra_thread_info;
  t->to_pid_to_str = linux_kthread_pid_to_str;
  t->to_stratum = thread_stratum;
  t->to_magic = OPS_MAGIC;

  return t;
}
```



Task Awareness

```
qemu-system-arm -kernel ./zImage -dtb ./vexpress-v2p-ca15-tc1.dtb -M vexpress-a15 -smp 2 -m 1024 -append 'root=/dev/nfs nfsroot=10.0.2.2:/opt/root/armv7/,tcp,v3 rw
ip=dhcp mem=1024M raid=noautodetect rootwait console=ttyAMA0,38400n8 devtmpfs.mount=0'     -nographic -gdb tcp::32770
[          0.000000] Booting Linux on physical CPU 0x0
[          0.000000] Linux version 4.6.0-rc1 (kbingham@CookieMonster) (gcc version 5.2.1 20151010 (Ubuntu 5.2.1-22ubuntu1) ) #13 SMP Thu Mar 31 10:33:19 BST 2016
[          0.000000] CPU: ARMv7 Processor [412fc0f1] revision 1 (ARMv7), cr=10c5387d
[          0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[          0.000000] Machine model: V2P-CA15
[ ..... ]
```

threads now appear in the inferior

```
lkd/bin/arm-linux-gdb ./linux/vmlinux  -iex 'add-auto-load-safe-path ./linux' -ex 'target remote localhost:32770'
Remote debugging using localhost:32770

(gdb) info threads
  Id   Target Id       Frame
* 1         [swapper/0] (TGID:0 <C0>) cpu_v7_do_idle () at ../linux/arch/arm/mm/proc-v7.S:74
  2         [swapper/1] (TGID:0 <C1>) cpu_v7_do_idle () at ../linux/arch/arm/mm/proc-v7.S:74
  3         init (TGID:1)         context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  4         [kthreadd] (TGID:2) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  5         [ksoftirqd/0] (TGID:3) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  6         [kworker/u4:0] (TGID:6) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  7         [rcu_sched] (TGID:7) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  8         [rcu_bh] (TGID:8) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  9         [migration/0] (TGID:9) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  10        [watchdog/0] (TGID:10) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
  11        [cpuhp/0] (TGID:11) context_switch (next=<optimized out>, prev=<optimized out>, rq=<optimized out>) at ../linux/kernel/sched/core.c:2734
[...]
```

Linaro

Extending GDB with Python

- Commands
- Functions
- Pretty Printing objects
- Frame Filters / Unwinders
- Breakpoints
- … and more ...

https://sourceware.org/gdb/onlinedocs/gdb/Python-API.html

Thread awareness in Python

```
def to_update_thread_list(self):
        inferior = gdb.selected_inferior()
        threads = inferior.threads()

        for task in tasks.task_lists():
            ptid = [inferior.pid, int(task['pid']), 0]  # (pid, lwp, tid)

            if ptid not in threads:
                    thread = inferior.new_thread(ptid, task)
                    thread.name = task['comm'].string()
                    # Provide thread registers for backtrace
                    self.setup_threads(thread, task)
```

Module Symbol support

kbingham@CookieMonster: ~

File Edit View Search Terminal Help

```
root@10.0.2.15:~# depmod -a
[   63.643135] random: nonblocking pool is initialized
root@10.0.2.15:~# modprobe helloworld
WARNING: All config files need .conf: /etc/modprobe.
d/invalid, it will be ignored in a future release.

[   73.866004] <1>Hello World 0 !
[   73.893862] Wow... kernel level thread saying hello :) : 0
[   73.924062] Wow... kernel level thread saying hello :) : 1
[   73.952099] Wow... kernel level thread saying hello :) : 2
```

kbingham@CookieMonster: ~

File Edit View Search Terminal Help

```
(gdb) lx-symbols /opt/rootfs/armv7/lib/modules/4.6.0-rc1/
loading vmlinux
(gdb) c
Continuing.

scanning for modules in /opt/root/armv7/lib/modules/4.6.0-
rc1/
scanning for modules in /home/lkd/targets/qemu-arm
loading @0xbf000000: ../lib/modules/4.6.0-
rc1/extra/helloworld.ko
```

Linux GDB extensions in v4.6-rc1

```
(gdb) apropos lx
function lx_current -- Return current task
function lx_module -- Find module by name and return the module variable
function lx_per_cpu -- Return per-cpu variable
function lx_task_by_pid -- Find Linux task by PID and return the task_struct variable
function lx_thread_info -- Calculate Linux thread_info from task variable
lx-cmdline --  Report the Linux Commandline used in the current kernel
lx-dmesg -- Print Linux kernel log buffer
lx-list-check -- Verify a list consistency
lx-lsmod -- List currently loaded modules
lx-ps -- Dump Linux tasks
lx-symbols -- (Re-)load symbols of Linux kernel and currently loaded modules
lx-version --  Report the Linux Version of the current kernel
```

Extending GDB with Python

gdb.Command : lx-cmdline

```
class LxCmdLine(gdb.Command):
  """ Report the Linux Commandline used in the current kernel.
        Equivalent to cat /proc/cmdline on a running target"""

  def __init__(self):
        super(LxCmdLine, self).__init__("lx-cmdline", gdb.COMMAND_DATA)

  def invoke(self, arg, from_tty):
        gdb.write(gdb.parse_and_eval("saved_command_line").string() + "\n")

LxCmdLine()
```

Extending GDB with Python

gdb.Command : lx-cmdline

```
class LxCmdLine(gdb.Command):
    """ Report the Linux Commandline used in the current kernel
```

```
(gdb) lx-cmdline
root=/dev/nfs nfsroot=10.0.2.2:/opt/root/armv7/,tcp,v3 rw ip=dhcp mem=1024M
raid=noautodetect rootwait console=ttyAMA0,38400n8 devtmpfs.mount=0

(gdb) help lx-cmdline
 Report the Linux Commandline used in the current kernel.
      Equivalent to cat /proc/cmdline on a running target
```

```
            gdb.write(gdb.parse_and_eval("saved_command_line").string() + "\n")


LxCmdLine()
```

Extending GDB with Python

gdb.Function : lx_task_by_pid

```
class LxTaskByPidFunc(gdb.Function):
    """Find Linux task by PID and return the task_struct variable.

$lx_task_by_pid(PID): Given PID, iterate over all tasks of the target and
return that task_struct variable which PID matches."""

    def __init__(self):
                super(LxTaskByPidFunc, self).__init__("lx_task_by_pid")

    def invoke(self, pid):
        task = get_task_by_pid(pid)
        if task:
                return task.dereference()
        else:
                raise gdb.GdbError("No task of PID " + str(pid))

LxTaskByPidFunc()
```

Extending GDB with Python

gdb.Function : lx_task_by_pid

```
class LxTaskByPidFunc(gdb.Function):
    """Find Linux task by PID and return the task_struct variable.
```

```
(gdb) lx-ps
0xeeea5500 1163 lircd                    ## Output trimmed ….
(gdb) set $task = $lx_task_by_pid(1163)
(gdb) print $task.comm
$5 = "lircd\000"
(gdb) print $task. <tab completion available>
```

```
LxTaskByPidFunc()
```

Extending GDB with Python

gdb.Function : lx_radix_tree_lookup (not in ML)

```
(gdb) print irq_desc_tree
$1 = {
  height = 1,
  gfp_mask = 37748928,
  rnode = 0xee000001
}
(gdb) print *irq_desc_tree.rnode
$2 = {
  path = 855638016,
  count = 0,
  ….
  slots = {0xc0ee8030, 0x80ee8030, 0x40ee8031, 0xee8032, 0xc0ee8033, ……
```

Extending GDB with Python

gdb.Function : lx_radix_tree_lookup (not in ML)

```
class LxRadixTree(gdb.Function):
    """ Lookup and return a node from a RadixTree.

$lx_radix_tree_lookup(root_node [, index]): Return the node at the given index.
If index is omitted, the root node is dereferenced and returned."""

    def __init__(self):
            super(LxRadixTree, self).__init__("lx_radix_tree_lookup")

    def invoke(self, root, index=0):
            result = lookup(root, index)
            if result is None:
                    raise gdb.GdbError("No entry in tree at index {}".format(index))

            return result

LxRadixTree()
```

[PATCHv4 10/12] scripts/gdb: Add a Radix Tree Parser
https://lkml.org/lkml/2016/3/30/277

Extending GDB with Python

gdb.Function : lx_radix_tree_lookup (not in ML)

```
(gdb) print ((struct irq_desc)$lx_radix_tree_lookup(irq_desc_tree, 18)).irq_data
$3 = {
  mask = 0,
  irq = 18,
  hwirq = 27,
  common = 0xee803d80,
  chip = 0xc100285c <gic_data>,
  domain = 0xee808000,
  parent_data = 0x0,
  chip_data = 0xc100285c <gic_data>
}
```

Extending GDB with Python : Accessing data

- GDB provides accessors to read memory

```
def module_list():
        modules = gdb.parse_and_eval("modules")
        entry = modules['next']
        end_of_list = modules.address
```

- Reading structures is ‘easy’

```
for vfs in lists.list_for_each_entry(namespace['list'],mount_ptr_type, "mnt_list"):
        devname = vfs['mnt_devname'].string()
        superblock = vfs['mnt']['mnt_sb']
        fstype = superblock['s_type']['name'].string()
        s_flags = int(superblock['s_flags'])
        m_flags = int(vfs['mnt']['mnt_flags'])
```

- Complicated data structures can be programmed

Python Extension Summary

- Easy to write your own commands / plugins to GDB
- Docstring as Documentation
- Accessing data in Python is easy
  - Structures organised as python dictionaries
  - Pointers automatically dereferenced

What's Next

- Automated regression testing
  - LAVA / KernelCI ...
- Continue upstream push of thread awareness
  - C / Python ~~/ Javascript~~

And then ?

- IDE integration
- Userspace debug extensions?
- Page table walks?
- The world ...

Summary

Targets

- KGDB
  - In kernel debugging
- QEmu / KVM / UML
  - Virtualized environments
- JTAG
  - Real Hardware
- Core Dumps
  - Real problems

Kernel Awareness

- Thread Awareness
  - In Progress!
- Module support
  - Mostly there
- Data Retrieval
  - Commands available
- Oysters or Pythons?
  - The world is your ...

Some references / Credits

- Linaro
  - http://www.linaro.org
- O'Rly Images
  - Buy the T-Shirts @ https://threddit.com/ThePracticalDev
- GDB Python API Documentation
  - https://sourceware.org/gdb/onlinedocs/gdb/Python-API.html
- Me
  - http://www.kieranbingham.co.uk

Slides should be available on ELC website, or from my Blog URL

Code/GIT URL's

- [PATCHv4 00/12] gdb/scripts: Linux awareness debug commands
  - https://lkml.org/lkml/2016/3/30/269
- Linux
  - https://git.linaro.org/people/kieran.bingham/linux.git
    - Tag: gdb-scripts-v4 - Latest submission
    - Branch: gdb-scripts - All work including experimental linux-awareness.py
- Binutils-GDB :
  - https://git.linaro.org/people/kieran.bingham/binutils-gdb.git/
    - Branch: lkd-thread-aware-c - Working version of thread awareness
    - Branch: linux-kthreads - Work in progress - C implementation for upstream
    - Branch: lkd-python - *Experimental* - Python gdb.Target
- Qemu Quickstart (to try thread awareness, using lkd-thread-aware-c)
  - git clone https://git.linaro.org/people/kieran.bingham/qemu-kernel.git
  - Make # builds kernel, and binutils-gdb
  - Terminal 1: make qemu-run | Terminal 2: make qemu-gdb

Q+A?

Linaro