GDB Linux Kernel Awareness

Peter Griffin
Linaro Member Services

“*Engineers and Devices Working Together*”

… with free & open source tools!

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Overview

- Rationale for GDB Linux Kernel Awareness
- Targets for debugging Linux with GDB
- Upstream GDB to debug the Linux kernel
- What is the ST landing team doing?
- What is “Linux awareness”?
- Approach Taken
- Progress so far
- Next steps
- Alternative approaches
- Demo
- Q&A

Rationale for GDB Linux Kernel Awareness

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

ST LT Mission

To provide a better debugging experience when using upstream GDB to debug Linux kernel via JTAG / OpenOCD, or QEMU.

Ways to Debug Linux Kernel with GDB

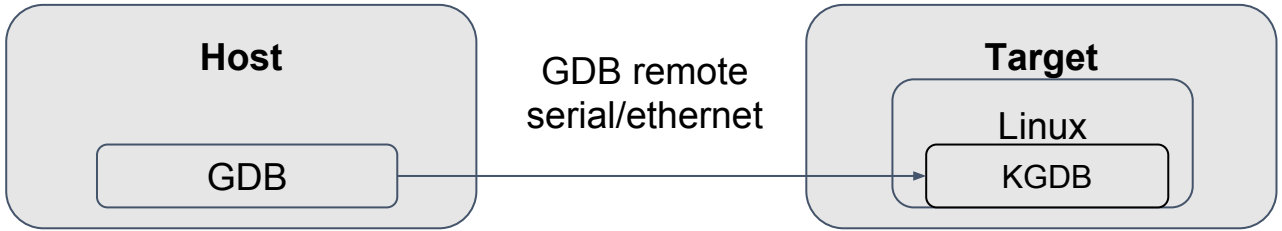
- GDB client using the GDB remote protocol
  a. Connection to a KGDB stub in a running kernel

  b. Connect to a QEMU stub running a virtual kernel environment
  c. Connect to a GDB remote 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 GDB remote protocol
  - Enable with CONFIG_KGDB
- \+ Already supported on many platforms
- \+ All kernel threads enumerated in GDB (via GDB remote packets)
- \- 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 and GDB

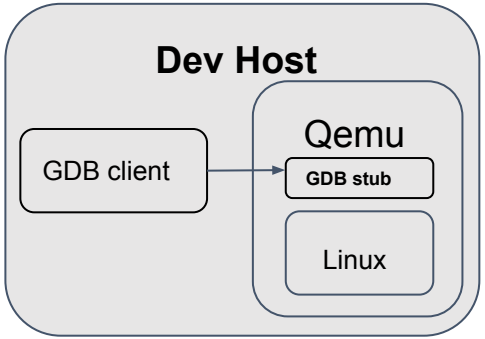
Advantages

+ QEMU is open source and has GDB remote stub
+ No “real” hardware required
+ Good for testing generic kernel code, on many architectures
+ **Good environment for developing GDB Linux awareness extensions**

Disadvantages

- If your bug is SoC or board related it is unlikely to be useful

.



QEMU and GDB (Example)

```
> qemu-system-arm -M vexpress-a9 -cpu cortex-a9 -m 256M -nographic -kernel ./zImage -append 'console=ttyAMA0,115200
rw nfsroot=10.0.2.2:/opt/debian/wheezy-armel-rootfs,v3 ip=dhcp' -dtb ./vexpress-v2p-ca9.dtb -s
[        0.000000] Booting Linux on physical CPU 0x0
[        0.000000] Linux version 4.3.0-rc3-00008-g6e751b6-dirty (griffinp@X1-Carbon) (gcc version 4.9.3 20141031
(prerelease) (Linaro GCC 2014.11) ) #317 SMP Tue Sep 29 13:20:36 BST 2015
[        0.000000] CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
[        0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[        0.000000] Machine model: V2P-CA9
[..]
[   98.288845] VFS: Unable to mount root fs via NFS, trying floppy.
[   98.290767] VFS: Cannot open root device "(null)" or unknown-block(2,0): error -6
[   98.291132] Please append a correct "root=" boot option; here are the available partitions:
[   98.291681] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(2,0)
[   98.292215] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 4.3.0-rc3-00008-g6e751b6-dirty #317
```

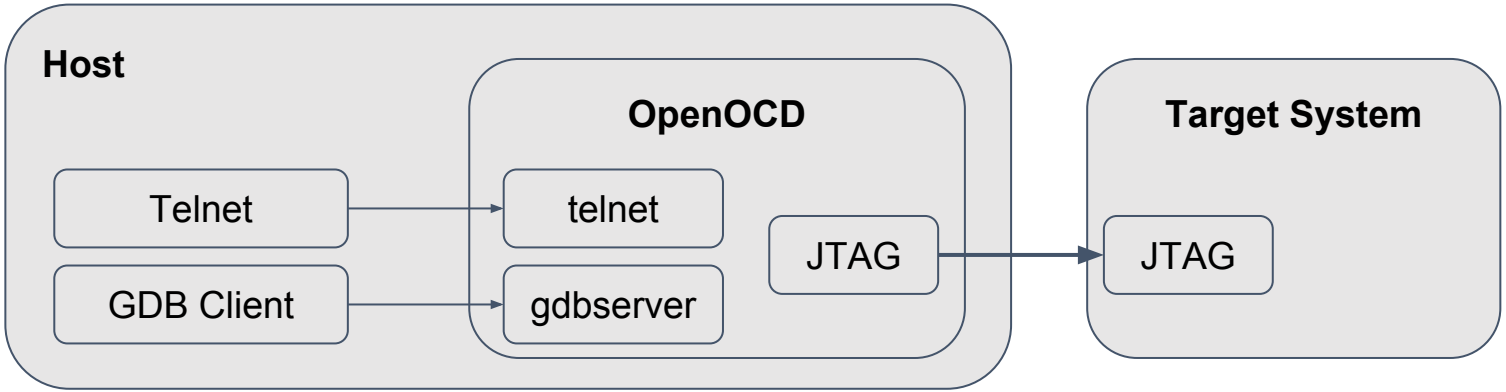
```
arm-linux-gnueabi-gdb vmlinux
(gdb) target remote :1234
Remote debugging using :1234
__loop_delay () at ../arch/arm/lib/delay-loop.S:47
47                subs    r0, r0, #1
(gdb) info threads
  Id   Target Id          Frame
* 1      Thread 1 (CPU#0 [running]) __loop_delay () at ../arch/arm/lib/delay-loop.S:47
(gdb) bt
#0  __loop_delay () at ../arch/arm/lib/delay-loop.S:47
#1  0xc02cbbe0 in panic (fmt=0xc0c340a8 "VFS: Unable to mount root fs on %s") at ../kernel/panic.c:201
#2  0xc0dad19c in mount_block_root (name=0xc0c34148 "/dev/root", flags=32768) at ../init/do_mounts.c:421
#3  0xc0dad354 in mount_root () at ../init/do_mounts.c:541
#4  0xc0dad4b4 in prepare_namespace () at ../init/do_mounts.c:600
#5  0xc0dace6c in kernel_init_freeable () at ../init/main.c:1026
#6  0xc09ca2f4 in kernel_init (unused=<optimised out>) at ../init/main.c:936
```

Targets: JTAG - External Debug

+ OpenOCD is open source
+ Supports GDB remote protocol
+ Supports many ARM/MIPS CPUs
+ Supports many FTDI based cheap JTAG probes

http://openocd.org

http://elinux.org/JTAG



Targets: JTAG - Why External Debugging?

- Allows external access to dedicated SoC debug hardware
- Flow control via on-chip hardware breakpoints and watchpoints
  - Access to on-chip performance monitors
  - Control reset and initialization of target
- Download code directly into RAM or flash
- Manipulate MMU and caches whilst target is halted
- Allows debugging when other techniques can’t be used e.g.
  - Ability to load code on bare board & debug from reset vector
  - Debug during late and early parts of system suspend & resume
  - Debug IRQ handlers
  - Inspect very broken / corrupt systems e.g. when KGDB stub no longer functional.

GDB Remote Compliant JTAG - e.g. OpenOCD

Compile it

```
> git clone  ssh://git@git.linaro.org:/people/peter.griffin/openocd-code
> cd openocd-code
> git submodule update --init --recursive
> autoreconf -iv
> export PKG_CONFIG_PATH=/usr/lib/x86_64-linux-gnu/pkgconfig/
> ./configure --enable-ftdi
> make
> make install
```

Run it

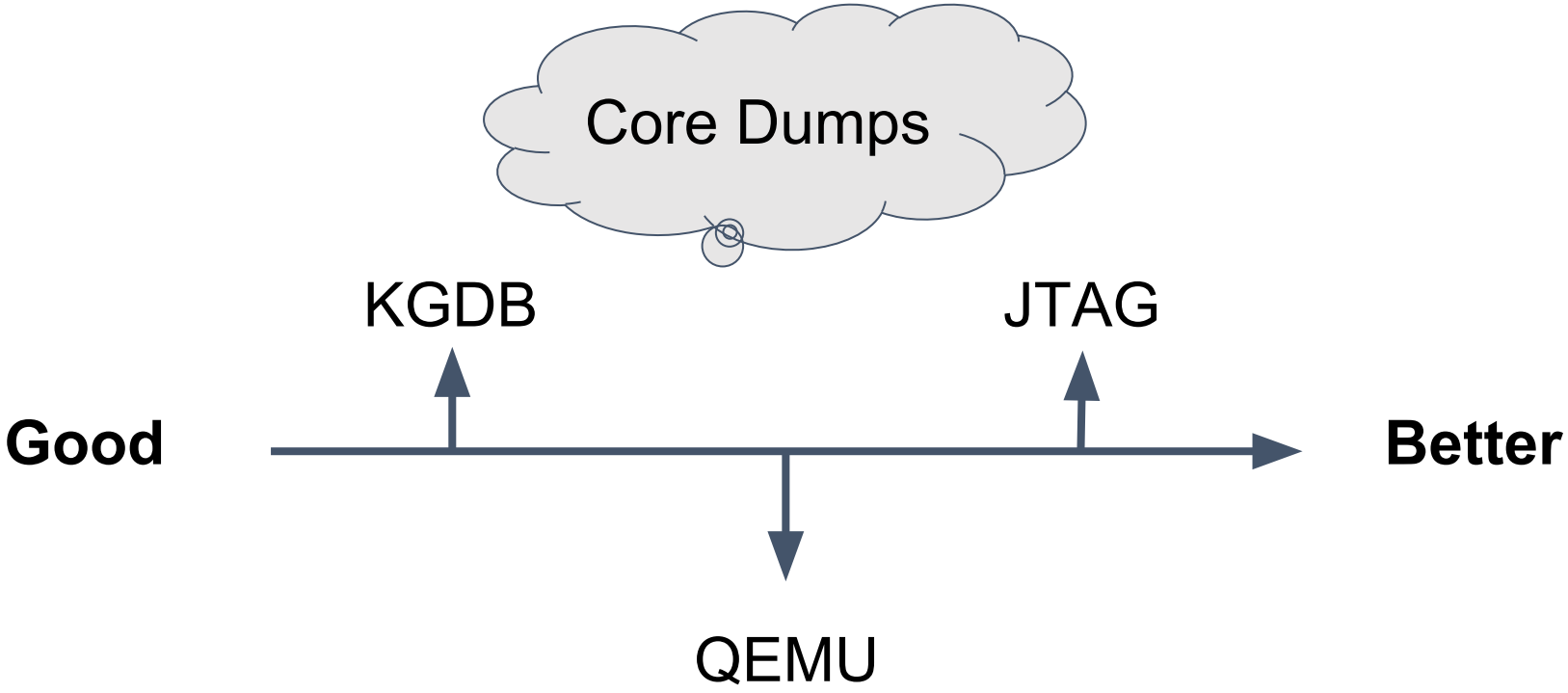
```
./openocd -f ./tcl/interface/ftdi/flyswatter2.cfg -f ./tcl/target/hi6220.cfg
Open On-Chip Debugger 0.9.0-dev-00241-gd356c86-dirty (2015-07-10-08:20)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.sourceforge.net/doc/doxygen/bugs.html
adapter speed: 10 kHz
Info : session transport was not selected, defaulting to JTAG
jtag_ntrst_delay: 100
trst_and_srst combined srst_gates_jtag trst_push_pull srst_open_drain
connect_deassert_srst
hi6220.cpu
Info : clock speed 10 kHz
Info : JTAG tap: hi6220.dap tap/device found: 0x5ba00477 (mfg: 0x23b,
part: 0xba00, ver: 0x5)
Info : hi6220.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Example telnet interface

```
telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
number of cache level 2
cache l2 present :not supported
hi6220.cpu cluster 0 core 0 multi core
target state: halted
target halted in ARM64 state due to debug-request,
current mode: EL2H
cpsr: 0x800003c9 pc: 0x3ef7e908
MMU: disabled, D-Cache: disabled, I-Cache: disabled
```

Example gdb interface trace

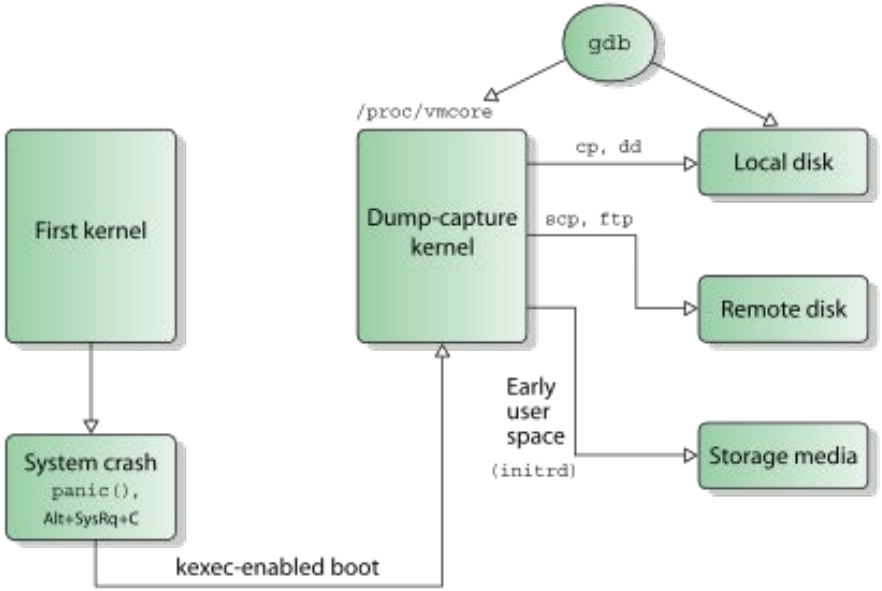
```
> aarch64-linux-gnu-gdb
(gdb) target remote :3333
(gdb) add-symbol-file build-hikey/u-boot 0x3ef47000
(gdb) hbreak do_version
Hardware assisted breakpoint 1 at 0x3ef4cc80: file
../common/cmd_version.c, line 19.
(gdb) c
Continuing.
Breakpoint 1, do_version (cmdtp=0x3ef8c0c0, flag=0,
argc=1, argv=0x3e746850) at ../common/cmd_version.c:19
19  {
(gdb) bt
#0  do_version (cmdtp=0x3ef8c0c0, flag=0, argc=1,
argv=0x3e746850) at ../common/cmd_version.c:19
#1  0x000000003ef604a8 in cmd_call (argv=0x3e746850,
argc=1,
```


Core Dumps
KGDB
JTAG
Good
Better
QEMU

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

Upstream GDB to Debug the Linux Kernel

So you have upstream GDB debugging the Linux Kernel via JTAG, or QEMU…

but compared to $$$ commercial tools or ST GDB it can be disappointing...

Upstream GDB to Debug the Linux Kernel

- **(gdb) info threads** - Shows thread running **on each physical CPU**
- No visibility of other sleeping threads
- No visibility of user processes


```
arm-linux-gnueabi-gdb vmlinux
(gdb) target remote :1234
Remote debugging using :1234
__loop_delay () at ../arch/arm/lib/delay-loop.S:47
47              subs    r0, r0, #1
(gdb) info threads
  Id   Target Id      Frame
* 1  Thread 1 (CPU#0 [running]) __loop_delay () at
../arch/arm/lib/delay-loop.S:47
```

Upstream GDB to Debug the Linux Kernel

- **(gdb) bt -** Works for CPU executing in kernel space. No unwinding for user tasks.

Example 'bt' executing in kernel space

```
arm-linux-gnueabi-gdb vmlinux
(gdb) target remote :1234
(gdb) bt
#0  __loop_delay () at
../arch/arm/lib/delay-loop.S:47
#1  0xc02cbbe0 in panic (fmt=0xc0c340a8
"VFS: Unable to mount root fs on %s") at
../kernel/panic.c:201
#2  0xc0dad19c in mount_block_root
(name=0xc0c34148 "/dev/root", flags=32768)
at ../init/do_mounts.c:421
#3  0xc0dad354 in mount_root () at
../init/do_mounts.c:541
#4  0xc0dad4b4 in prepare_namespace () at
../init/do_mounts.c:600
```

Example 'bt' executing in userspace

```
Program received signal SIGINT,
Interrupt.
0xb6e4978c in ?? ()
(gdb) bt
#0  0xb6e4978c in ?? ()
#1  0xb6e4a408 in ?? ()
Backtrace stopped: previous frame
identical to this frame (corrupt
stack?)
```

Upstream GDB to Debug the Linux Kernel

`up / down / frame <num>` – same as bt, will show physical CPU, and only work if executing in kernel space.

`thread <num>` – switch physical CPUs

This could be better, more like $$$ proprietary JTAG tools.

What is Linux Awareness?

- Provide the debugger with additional knowledge of the underlying O/S to enable a better debugging experience.
  - Where is the Task List?
  - Where 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 GDB context commands (`info threads`, `thread <num>`, `frame <num>`, `up`, `down`)
  2. Loadable Module Support
     - Hooks for automatic symbol resolution when modules are inserted
  3. OS Helper Commands
     - Interacting with the debugger to extract state from the O/S

What is the ST Landing Team Doing?

- Working to extend Linux Kernel Awareness in upstream GDB.
  - With support from the Linaro Toolchain Working Group.
- This is based on the C Linux Kernel Debugger (LKD) GDB plugin by ST; the aim now is to extract and rework the features that are generally useful, and upstream them.

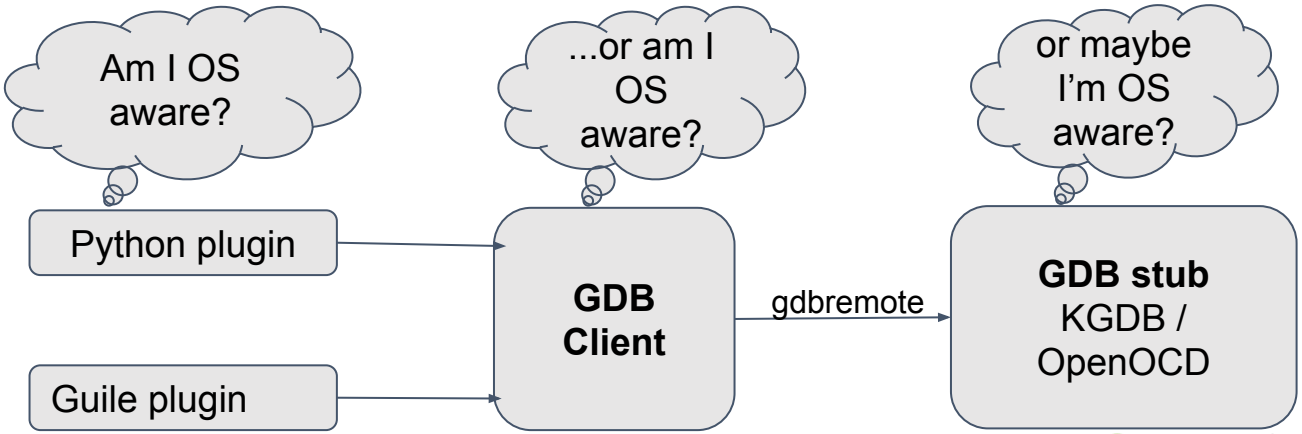
- The team has engaged with the GDB and Linux kernel communities to develop an approach.
- ST and the Linaro ST Landing Team are publicising this work to other Linaro members who might have an interest in it.

Overview of ST LKD C Patchset

1129	lkd-arm.c	**ARM arch specific** (virt to phys)
674	lkd.h	
491	lkd-irqs.c	
4953	lkd-main.c	**linux-aware target_ops, OS helpers**
2456	lkd-modules.c	**modules target_so_ops**
79	lkd-modules.h	
1666	lkd-process.c	**kthread awareness**
77	lkd-process.h	
220	lkd-proxy.c	
2076	lkd-sh4.c	SH4 specific
13874	**total**	

Where to Put the “Awareness”?

- We have 3 options
  1. Scripting extension to GDB (Python / Guile)
  2. Awareness in GDB stub
  3. C extension to GDB
- Each of these approaches has advantages and disadvantages



Extending GDB with Python

- You can extend GDB using Python. See: https://sourceware.org/gdb/onlinedocs/gdb/Python.html#Python

- What is exposed via Python interface has been steadily improving in GDB
  - Add custom commands
  - Implement pretty printers
  - Frame filters, frame decorators and much more.

- The **major** advantage of implementing “Linux awareness” in Python is: the code can live in the kernel source tree.
  + **Extensions should evolve with the kernel source**
  + No cross-dependencies between kernel & GDB

Python GDB Linux Awareness “Extensions”

Jan Kiszka from Siemens has already implemented some Python Linux awareness extensions and established this route.

- See Docs @ Documentation/gdb-kernel-debugging.txt
- See Python @ scripts/gdb/*
- Enable CONFIG_GDB_SCRIPTS
- Provides a series of custom Linux commands and functions
- Can be easily tested using QEMU or OpenOCD

Linux GDB Python Extensions in v4.6

(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

ST LT Involvement: OS Helpers

- Migrating LKD C-based OS helper commands into Python.
  - C code then retired from LKD, reducing amount of 'out of tree' LKD code.
  - Python code when merged will evolve with the kernel source tree.
- Some commands such as **dmesg** already existed upstream
- First ST LT Python OS helper commands merged into Linux kernel.
  - lx-cmdline and lx-version in **v4.6**.
  - lx-mount, lx-iomem, lx-ioports in **v4.7**.

```
lx-cmdline --  Report the Linux Commandline used in the current kernel
lx-version --  Report the Linux Version of the current kernel
lx-iomem -- Identify the IO memory resource locations defined by the kernel
lx-ioports -- Identify the IO port resource locations defined by the kernel
```

Example Command Output

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

 (gdb) lx-version
Linux version 4.7.0-rc3-00003-g5d518fc-dirty
(griffinp@griffinp-ThinkPad-X1-Carbon-2nd) (gcc version 4.9.3 20141031 (prerelease)
(Linaro GCC 2014.11) ) #4 SMP Thu Jul 14 12:54:03 BST 2016
```

Example Output

```
(gdb) lx-iomem
  10000008-1000000b : dat
  10000048-1000004b : dat
  1000004c-1000004f : dat
  100000a0-100000ab : vexpress-syscfg
  10004000-10004fff : /smb@04000000/motherboard/iofpga@7,00000000/aaci@04000
  10005000-10005fff : /smb@04000000/motherboard/iofpga@7,00000000/mmci@05000
    10005000-10005fff : /smb@04000000/motherboard/iofpga@7,00000000/mmci@05000
  10006000-10006fff : /smb@04000000/motherboard/iofpga@7,00000000/kmi@06000
    10006000-10006fff : kmi-pl050
  10007000-10007fff : /smb@04000000/motherboard/iofpga@7,00000000/kmi@07000
    10007000-10007fff : kmi-pl050
  10009000-10009fff : /smb@04000000/motherboard/iofpga@7,00000000/uart@09000
    10009000-10009fff : /smb@04000000/motherboard/iofpga@7,00000000/uart@09000
  1000a000-1000afff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0a000
    1000a000-1000afff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0a000
  1000b000-1000bfff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0b000
    1000b000-1000bfff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0b000
  1000c000-1000cfff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0c000
    1000c000-1000cfff : /smb@04000000/motherboard/iofpga@7,00000000/uart@0c000
  10011000-10011fff : /smb@04000000/motherboard/iofpga@7,00000000/timer@11000
  10012000-10012fff : /smb@04000000/motherboard/iofpga@7,00000000/timer@12000
  10017000-10017fff : /smb@04000000/motherboard/iofpga@7,00000000/rtc@17000
    10017000-10017fff : rtc-pl031
  1001f000-1001ffff : /smb@04000000/motherboard/iofpga@7,00000000/clcd@1f000
    1001f000-1001ffff : clcd-pl11x
    [..]
```

lx-fdtdump (Not Upstreamed Yet)

```
lx-fdtdump -- Output Flattened Device Tree header and dump FDT blob to a file
```

```
(gdb) lx-fdtdump
fdt_magic:          0xD00DFEED
fdt_totalsize:      0xC108
off_dt_struct:      0x38
off_dt_strings:     0x3804
off_mem_rsvmap:     0x28
version:            17
last_comp_version: 16
Dumped fdt to fdtdump.dtb
```

Allows further post-processing on development machine with standard DT tools.

```
e.g. fdtdump fdtdump.dtb | less
```

Allows diagnoses of what bootloader may have changed in DT to cause board not to boot.

Kernel Thread Awareness: ST LT Approach

- Working to extract a minimal C-based kthread patchset for GDB.
- Minimal Linux kthread RFC posted
  https://sourceware.org/ml/gdb-patches/2016-02/msg00826.html
- Not much feedback from GDB community.

Now preparing a refactored C implementation of linux-kthread for upstreaming.

Based on feedback from Yao Qi at Connect BKK16

(GDB maintainer & ARM TCWG assignee to Linaro).

Fixing style issues, re-factoring architecture specifics etc.

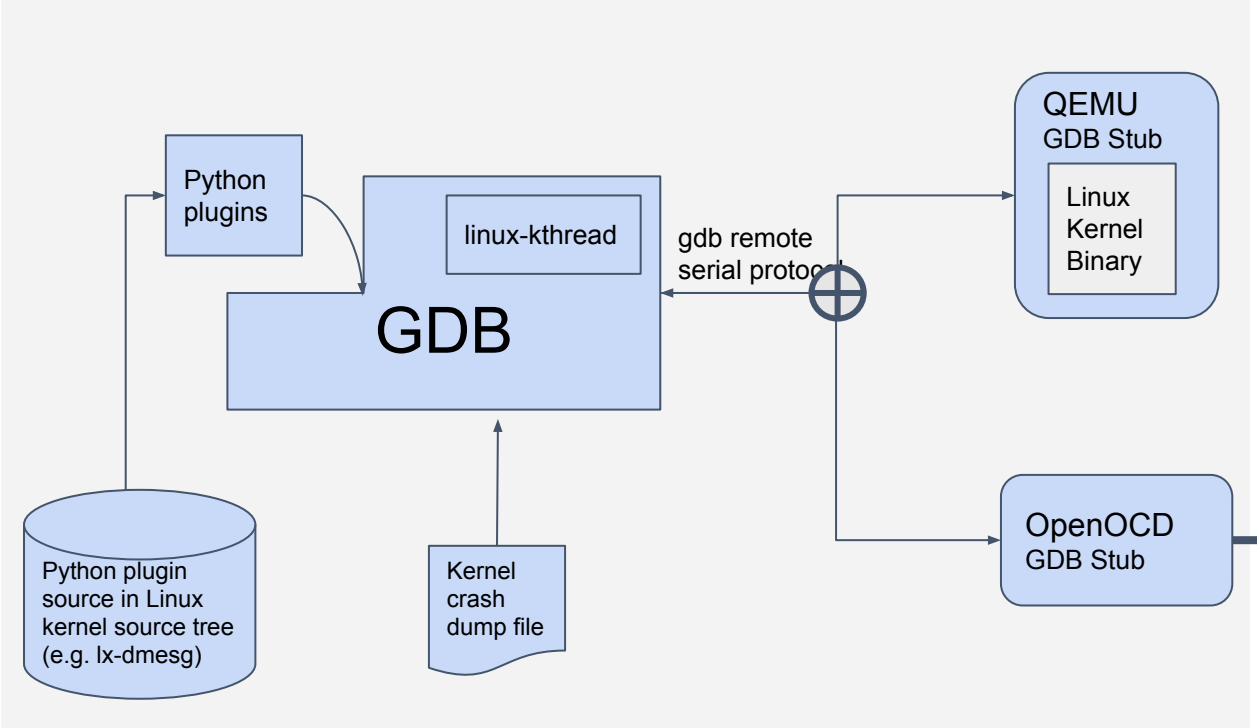
Patches work well with nommu Linux, OpenOCD and GDB.

Aim to submit refactored patches to GDB community by end of Q4 2016.

Overview linux-kthread Refactored Patchset

```
 ~/binutils-gdb/gdb$ git diff 4f80d1d --stat
 gdb/Makefile.in        |    4 +-
 gdb/arm-linux-tdep.c   |   98 ++++++
 gdb/configure.tgt      |    6 +-
 gdb/doc/observer.texi  |    5 +
 gdb/linux-kthread.c    | 1548
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
 gdb/linux-kthread.h    |  215 +++++++++++++
 gdb/remote.c           |    5 +
 gdb/thread.c           |    2 +-
 8 files changed, 1878 insertions(+), 5 deletions(-)
```

GDB Linux Kernel Awareness: Components

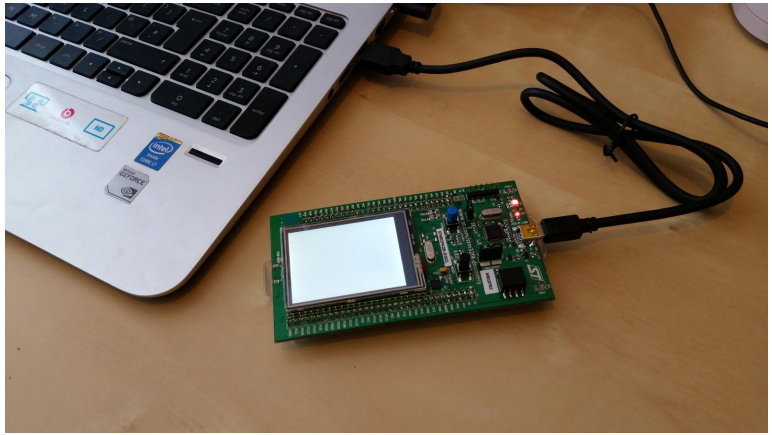


Demo!

- Refactored minimal kthread patchset
- OpenOCD controlling stm32 target
- STM32F429 Discovery Board running v4.7-rc3 **nommu** Linux

Demonstrating:

- Correct enumeration of kernel threads
  - New threads enumerated by GDB
  - Old threads pruned from GDB thread list
- Python extensions merged for v4.7
  - lx-cmdline, lx-mounts, lx-iomem



Current Kthread Challenges and Considerations

Virtual Memory and MMU control

- User can halt CPU at anytime and MMU could be configured for userspace mapping.
  - Must be able to detect this, before attempting to read any kernel memory.
  - Updating kernel thread list in GDB requires us to to switch MMU to kernel mapping to read kernel memory and parse task_struct list.
- Prototyping with QEMU to return ARM system coprocessor registers over GDB remote using target XML.
  - Currently returns TTBR0/1, MIDR amd IDPFR0/1 registers.
  - Using target XML allows GDB stub to return appropriate registers needed for full system debug.
  - Using registers, allows access via register cache, and also from python scripts (unlike for example custom GDB remote packet).
  - Support for target XML currently not implemented in OpenOCD.

GDB knowledge of CPU core

- To determine whether virtual memory manipulation is required.

CP Registers Example

```
(gdb) info registers
r0             0x1         1
[..]
sp             0xc1201fa0  0xc1201fa0 <init_thread_union+8096>
lr             0xc03087e8  -1070561304
pc             0xc031b0e8  0xc031b0e8 <cpu_v7_do_idle+8>
cpsr           0x60000093  1610612883
ttbr0_el1      0x0 0
ttbr1_el1      0x6020406a  1612726378
midr           0x410fc090  1091551376
id_pfr0        0x1031      4145
id_pfr1        0x11 17
```

Next ST LT Steps

- Complete and upstream C implementation of linux-kthread.
- Automated regression testing (LAVA, CI).
- Implement and upstream remaining LKD OS helper commands
  - lx-pgtable, pgmap, process_info (all remaining commands rely on virtual memory support).
- Review and test upstream Python implementation of loadable module support versus LKD C implementation.

Known Bugs

- GDB APIs can behave differently between GDB remote stubs (e.g. QEMU versus OpenOCD stub)
- Bug in GDB Python shim layer https://bugs.linaro.org/show_bug.cgi?id=2489

Other Potential Extensions

- Alternatively could implement Linux kthreads in Python
  - Extend GDB by means of Python scripts. Python again lives in kernel source.
  - Need to extend GDB's Python API.
    - GDB thread object, target object, register object, symbol object need to be exposed via Python bindings to get working.
    - Infrastructure could be re-used for other OS's or RTOSs
  - Some prototyping and patches by ST LT to do this.
- Implement kthread for ARMv8 and X86 architectures.
  - Can be tested using QEMU, but needs also relies on aarch64 support in OpenOCD.
- Extend to support examination of kernel crash dump files.
  - Some GDB developers (in SUSE) working on this.
  - Could investigate how to combine this work with ours.

Future Extensions: Backtrace to Userspace!

- ST's LKD GDB plugin implements this feature.
  - Automatically pull and resolve userspace symbols for userspace processes from target root filesystem.

```
gdb) bt
#0  context_switch (cookie=..., next=<optimised out>, prev=<optimised out>, rq=<optimised out>) at ../kernel/sched/core.c:2867
[..]
#6  0xc042bd30 in do_select (n=<optimised out>, fds=<optimised out>, end_time=<optimised out>) at ../fs/select.c:533
#7  0xc042bee8 in core_sys_select (n=11, inp=<unavailable>, outp=<unavailable>, exp=0x0, end_time=0xee075f70) at
../fs/select.c:603
#8  0xc042c280 in SYSC_select (tvp=<optimised out>, exp=<optimised out>, outp=<optimised out>, inp=<optimised out>,
n=<optimised out>) at ../fs/select.c:644
#9  SyS_select (n=11, inp=-1098552816, outp=0, exp=<unavailable>, tvp=-1098553000) at ../fs/select.c:626
#10 ret_fast_syscall () at ../arch/arm/kernel/entry-common.S:37          <----------          Kernel / User boundary
#11 0xb6f19a80 in elf_machine_rel (reloc=0x86, version=<optimised out>, reloc_addr_arg=0x3091969, sym=0x9db0,
map=0xbe856960) at ../ports/sysdeps/arm/dl-machine.h:475
#12 elf_dynamic_do_rel (lazy=0, relsize=<optimised out>, reladdr=<optimised out>, map=0xbe856960) at do-rel.h:120
#13 _dl_start (arg=0xbe8ab000) at rtld.c:547
#14 0xbe8ab000 in ?? ()
#15 0xbe8ab000 in ?? ()
```

Q+A?

Linaro
connect
Las Vegas 2016

ENGINEERS
AND DEVICES
WORKING
TOGETHER

Thread Awareness in GDB Stub

OpenOCD -rtos Linux Task Awareness

- An example of putting “thread awareness” in the GDB stub
- OpenOCD already supports various rtos task awareness (eCos, ThreadX, FreeRTOS, ChibiOS, embKernel, mqx and Linux)!
  - See openocd-code/src/rtos/linux.c
  - Disabled by default
- Enabled by adding “-rtos linux” to the target command in OpenOCD config file
  - `target create $_TARGETNAME_2 cortex_a -chain-position $_CHIPNAME.dap -dbgbase $_DAP_DBG2 -coreid 1 -rtos linux`

- This has been tested with Tincantools’ Flyswatter JTAG and a U8500 (armv7) snowball board.
- Required some hacking to get it working, but in the end I could successfully enumerate a full thread list in GDB!

OpenOCD -rtos Task Awareness

Advantages

+ No GDB changes required (threads returned via gdbremote packets).
+ All kernel threads can be enumerated in GDB (once working)
+ Can be used in conjunction with python extensions (e.g. lx-dmesg etc)
+ OpenOCD already has mmu virt to phys translation code for many CPUs

Disadvantages

- **Awareness in GDB means thread awareness re-implementing for each stub: Qemu, OpenOCD...**
- **No good for debugging kernel dumps**
- **Tight coupling of code parsing kernel data structures in OpenOCD**
- **No debug information available in OpenOCD.**
- No way (I know of) to find field offsets via gdbremote protocol.
- Currently this means task_struct, thread_info and mm_struct field offsets require OpenOCD to be **recompiled** for the kernel you wish to debug :(

Improvements

- gdbremote protocol could be extended
- At a minimum task structure field offsets configurable at runtime, instead of requiring recompile!

OpenOCD linux-header.h

```
  /*  gdb script to update the header file  according to kernel version and build option
before executing function awareness  kernel symbol must be loaded : symbol vmlinux
  define awareness
   set logging off
   set logging file linux_header.h
   set logging on
   printf "#define QAT %p\n",&((struct task_struct *)(0))->stack
   set $a=&((struct list_head *)(0))->next
   set $a=(int)$a+(int)&((struct task_struct *)(0))->tasks
   printf "#define NEXT  %p\n",$a
   printf "#define COMM  %p\n",&((struct task_struct *)(0))->comm
   printf "#define MEM  %p\n",&((struct task_struct *)(0))->mm
   printf "#define ONCPU %p\n",&((struct task_struct *)(0))->on_cpu
   printf "#define PID %p\n",&((struct task_struct *)(0))->pid
  */
```

```
#define QAT 0x4
#define NEXT  0x1b0
#define COMM  0x2d4
#define MEM  0x1cc
#define ONCPU 0x18
#define PID 0x1f4
#define CPU_CONT 0x1c
#define PREEMPT 0x4
#define MM_CTX 0x160
```

Thank You

#LAS16

For further information: www.linaro.org

LAS16 keynotes and videos on: connect.linaro.org