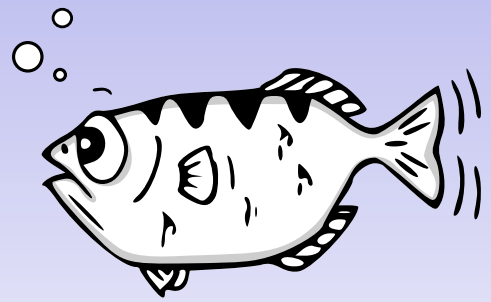


Debugging Linux Kernel Dumps with GDB?

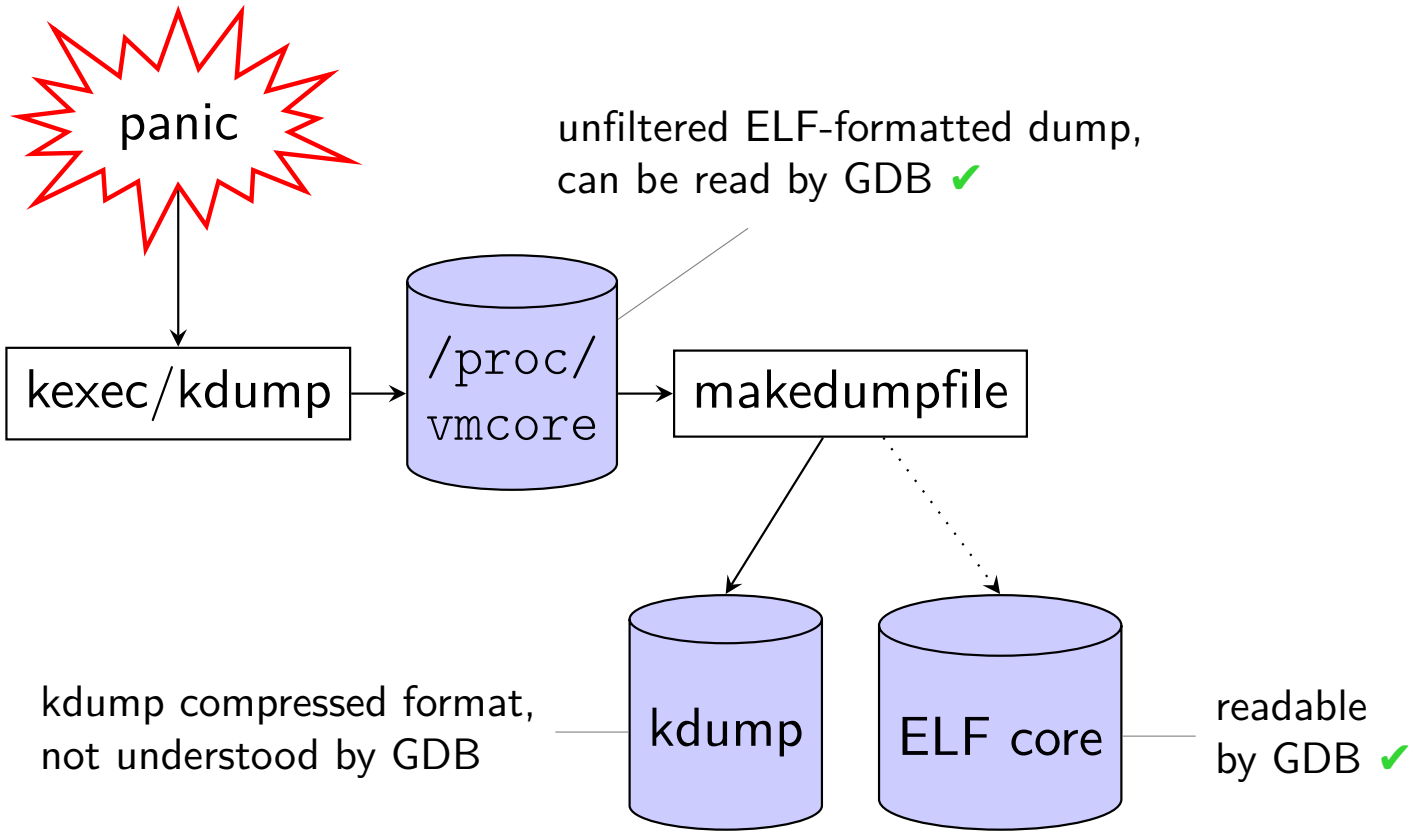
Andreas Arnez

August 8, 2015, GNU Tools Cauldron, Prague

IBM

GDB for Linux Kernel Debugging (1)

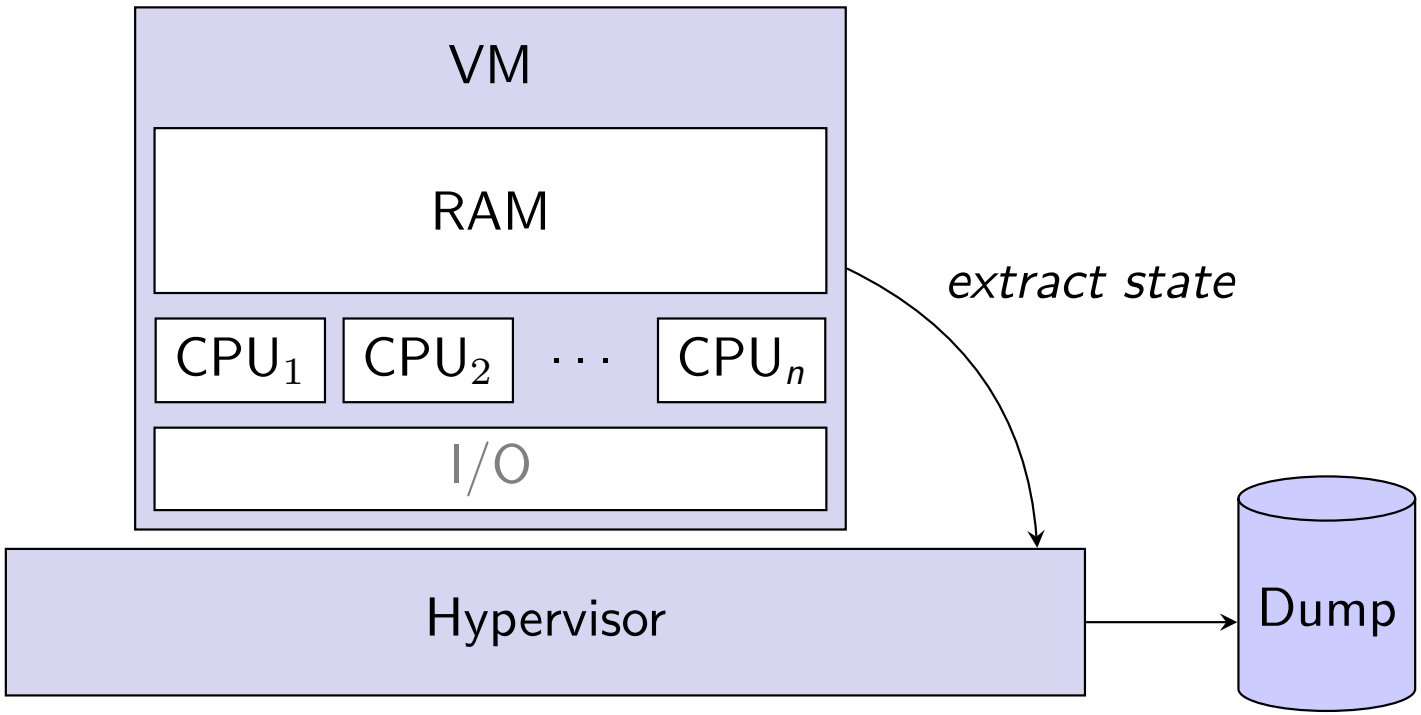
Dump via Kexec/Kdump




```
$ gdb /usr/src/linux/vmlinux dump.elf
```


GDB for Linux Kernel Debugging (2)

Dump Taken by Hypervisor



Hypervisor-created dumps often have a special format.

➔ Must be converted before analyzing them with GDB.

GDB for Linux Kernel Debugging (3)

/proc/kcore

`/proc/kcore` – ELF core file image of the running kernel.

```
$ sudo gdb /usr/src/linux/vmlinux /proc/kcore
```

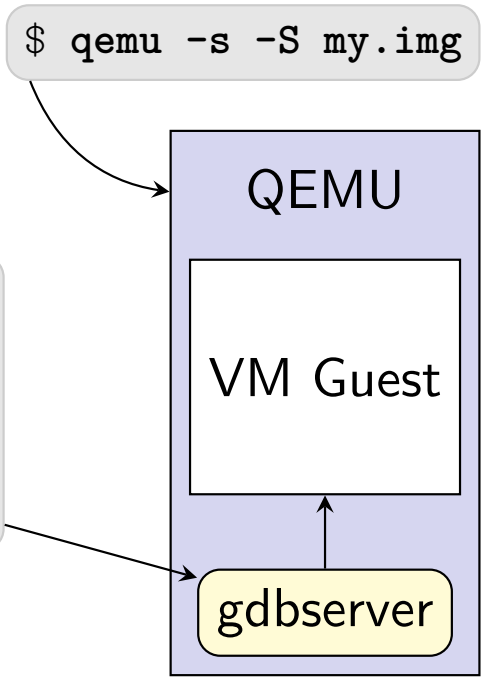
- Much like debugging a kernel crash dump.
- Can be done on a running system.
  - But not necessarily well-suited to observe things “in motion”.

GDB for Linux Kernel Debugging (4)

gdbserver in Qemu

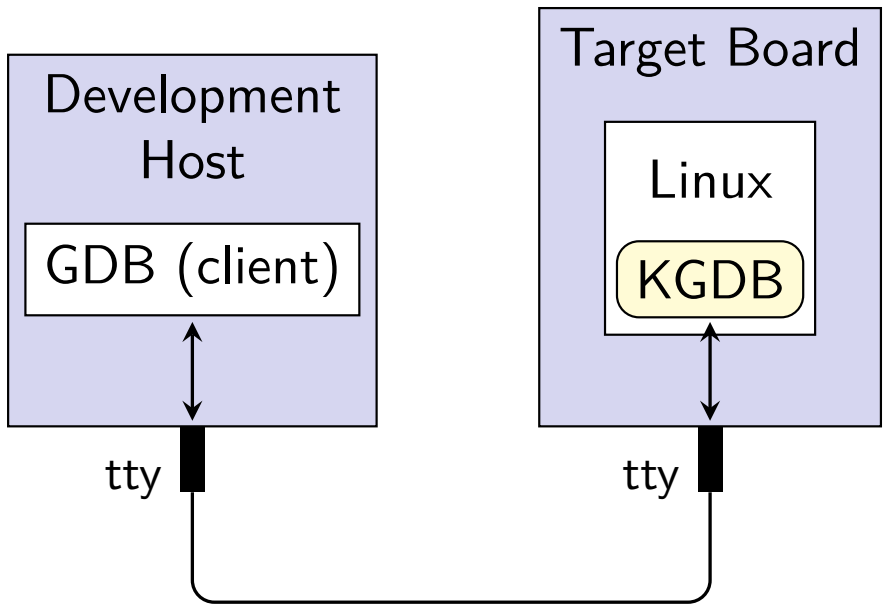
GDB (client):

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0x0000fff0 in ?? ()
(gdb) symbol-file vmlinux
Reading symbols from vmlinux...done.
```



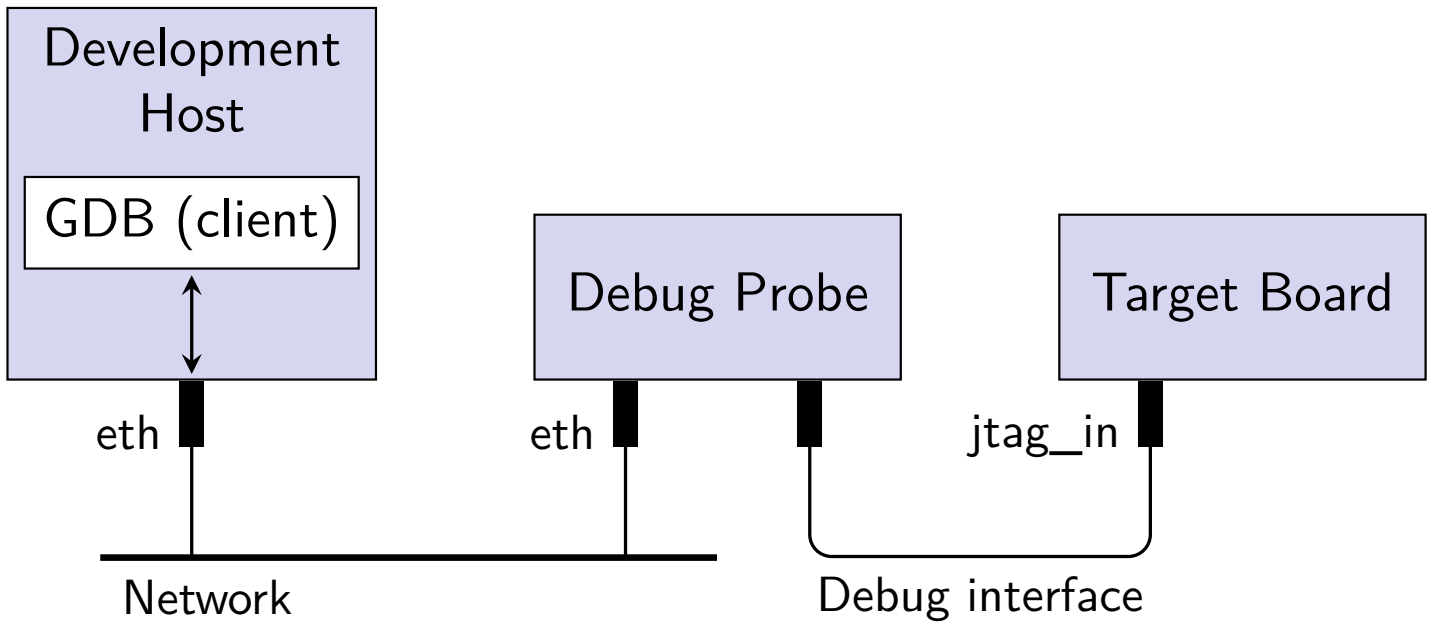
GDB for Linux Kernel Debugging (5)

KGDB



GDB for Linux Kernel Debugging (6)

Debug Probe (JTAG)



GDB Helper Scripts for Linux

Some GDB Python scripts are contained in the Linux source tree:

- Under `scripts/gdb`.
- See `Documentation/gdb-kernel-debugging.txt`.

Commands and functions provided:

`$lx_current()`	Return current task.
`$lx_module()`	Find module by name and return the module variable.
`$lx_per_cpu()`	Return per-cpu variable.
`$lx_task_by_pid()`	Find Linux task by PID and return the `task_struct` variable.
`$lx_thread_info()`	Calculate Linux `thread_info` from task variable.
`lx-dmesg`	Print Linux kernel log buffer.
`lx-lsmod`	List currently loaded modules.
`lx-symbols`	(Re-)load symbols of Linux kernel and currently loaded modules.

Crash Utility

The **crash** analysis utility is. . .

- inspired by the SVR4 UNIX `crash` command;
- mainly targeted at analyzing Linux kernel dumps;
- maintained by David Anderson (Red Hat).

It has:

- a text-based UI;
- its own set of commands;
- a copy of GDB (with modifications) built in;
- support for many dump formats.

1 Using GDB for Linux Kernel Debugging

2 Possible Improvements

Motivation

While gdb is an incredibly powerful tool, it is designed to debug user programs, and is not at all "kernel-aware".

– David Anderson, from "White Paper: Red Hat Crash Utility"

Virtual vs. Physical Addresses

Elf segment headers include the following fields:

		used by. . .	
Name	**description**	**GDB**	**Crash**
`p_vaddr`	virtual address	✔	✘
`p_paddr`	physical address	✘	✔

In general, GDB treats `vmlinux` *as if it was a user-space program.*

➔ Better have GDB perform the virt→phys translation?

- New command **`target kcore`** to differentiate?
  - Or auto-detection?

Kernel Address Space Mapping

For the most part, kernel addresses are mapped linearly.

But consider...

- ➔ `vmalloc`:
  - Dynamically creates a new, non-contiguous mapping.
  - Used for allocating kernel modules.
- ➔ the z/Architecture *low core*:
  - Bytes 0–8191 in each CPU's "real" address space.
  - Mapped to different physical location per CPU.

Compressed Kdump Files

Kdump format advantages

- Compression built in.
- Efficiently represents memory holes.
- `makedumpfile` generates it by default.

Possible support in GDB

➔ New command **`target kdump`**?

➔ Possibly exploit libkdumpfile:
`https://github.com/ptesarik/libkdumpfile`

Kernel Modules as Shared Libraries

User Space Program

E. g.: `(gdb) info shared` – Shows currently loaded shared libraries.

How does GDB do this?

- Walk dynamic loader's list of shared libraries.
- Add hooks to dynamic loader for refreshing the list.

Linux Kernel

Kernel modules are the kernel's "shared libraries".

➔ Support same level of functionality as for user space programs?

GDB Commands Related to Execution Contexts (1)

Current State

Meaning of GDB commands when debugging the Linux kernel:

`info threads`	Show CPUs.
`thread` *num*	Switch to CPU *num*.
`bt`	Only for CPUs executing in kernel space.
`up`/**`down`**/**`frame`** *n*	Likewise.
`info inferiors`	(One inferior only.)

- No direct access to
  - sleeping tasks;
  - interrupted tasks;
  - user space tasks.

GDB Commands Related to Execution Contexts (2)

Possible Improvements

Suggestions for improved behavior:

`info threads`	Show all tasks (running and sleeping).
`lx-cpus`	List CPUs and what they are executing.
`thread` *num*	Switch to task *num*.
`bt`	For the current task, but kernel space only. Unwind beyond interrupt handler (similar to signal handler).
`up`/`down`/`frame` *n*	Likewise.
`info inferiors`	(One inferior only.)

- Still no direct access to user space tasks.

GDB Commands Related to Execution Contexts (3)

Possible Improvements – Stage 2

`info inferiors`	Like `ps`, list processes. Include user space processes and a "kernel process".
`inferior` *num*	Switch to process *num*.
`info threads`	Show threads in the current process.
`lx-cpus`	List CPUs and what they are executing. Include inferior IDs.
`thread` *num*	Switch to thread *num* in the current inferior.
`bt`	For the current thread. Unwind beyond interrupt handler, but no transition between kernel-/user space.
`up`/**`down`**/**`frame`** *n*	Likewise.
`generate-core-file`	Write core dump for current (user-space) inferior.

C-Like Extension Language for GDB?

E. g., **Eppic** – a “C interpreter”:

- Provided as a library (see https://code.google.com/p/eppic/).
- Has many call backs, such as for accessing memory, providing type definitions, variables, etc.
- Used by crash and makedumpfile.

Possible use in GDB

➔ Alternative to `compile` when inferior has no running process.

➔ Yet another scripting language ☺