No Crash Dump? No Problem!

Light-weight remote kernel crash reporting for settop boxes

David VomLehn, Technical Leader

CELF 2010

Imagine…

It's the last five minutes of the Superbowl...

The game is tied...

And...

...2 million set top boxes with your company's logo crash!

Ouch!

To Prevent That, We Need Information

- Need to be able to have a good chance of figuring out what went wrong but, in the embedded world, we have constraints…
  - – Little persistent storage
  - – Tight memory
  - – Low or non-existent bandwidth
  - – Need to get the device doing it’s job quickly
  - – Analysis needs to be quick, so text reports, please

Crash Dumps Are Great!

- Very common method of capturing failure information
- With all of kernel memory available, there is a good chance of figuring out what went wrong
- Copy of system memory written to mass storage device, possibly with holes and compression to minimize required resources
- Supported in Linux with kexec/kdump on many/most/all(?) platforms

Crash Dumps-Not For Everyone

- Even using hole elimination and compression, core dumps are big
- Dumping all that memory takes time
- Need separate solution for userspace failures
- Just saving crash dump files prior to analysis can take a lot of space
- Not human readable and incomprehensible to those with limited expertise

What needs to be done?

- Report – Generate and store data to use for analysis
- Reboot – Re-initialize the system so that devices are available
- Send – Send the report upstream (or store locally)

Report Excerpts

- Custom header

```
DIAG 0x03: Kernel Diagnostic Report: 2.0 (Linux)
DIAG 0x03: Reason: Fatal exception in interrupt
DIAG 0x03: RF MAC: 00:22:CE:71:FF:BC
DIAG 0x03: HW MODEL: 335, HW VER: 14
DIAG 0x03: Uptime: 0d:20h:46m:58s
DIAG 0x03: Linux Kernel Release: 2.6.24-1.2.25.101_full-highmem
DIAG 0x03: Linux Kernel Version: #0 PREEMPT Sat Jun 27 13:43:01 PDT 2009
```

- Standard Registers, printed with printk()

```
Cpu 0
$ 0   : 00000000 10000301 cccccccc 9efe5ea0
$ 4   : 9121848c 90845c24 00000101 00000101
```

- Log extract

```
 | [66090.760000] pmem: kPmem_IoctlCmdGet 1074556930
 | [66090.760000] pmem: ioctl returning stst = 0
 | [69690.159000] pmem: kpmem_Ioctl  400c7002
 | [69690.159000]
 | [69690.159000] pmem: kPmem_IoctlCmdGet 1074556930
```

- Much more…

```
Kernel Memory Statistics
------------------------
MemTotal:        468368 kB
MemFree:         370348 kB
Buffers:          30168 kB
Cached:           31852 kB
```

RRSR - Report/reboot/send/reboot

- Proposed by Eric Biederman
- Boots light-weight kernel (kdump), which doesn't really need to be Linux, without reset
- Pros:
  - Can write failure report to any device, even send upstream via network
  - Very flexible
- Cons:
  - Dedicated memory need for kdump
  - Need real kernel for full device operability, scripts, etc.
  - Not working when writing failure report from memory
  - Additional storage for light-weight kernel
  - Not available until after boot complete

RRS - Report/reboot/send

- When failure occurs, generate and write report to vlram (pseudo-device that uses RAM preserved during reboots)
  - In kernel space, call vlram_panic_write()
  - In user space, write() to /dev/mtd0
- Reboot system
- Read from /dev/vlram0
  - Send report upstream
  - Store report locally

Report/reboot/send, II

- Pros
  - Gets system working again ASAP
  - Failure report sending done in parallel with saving report or sending upstream
  - Though it requires dedicated vlram memory during reboot, it can be reused after report captured or sent
  - Can start sending in parallel to bringing up the user space application
  - Provides userspace reporting, too
- Cons
  - Risk of losing report during reboot due to memory corruption, multiple failures, etc.
  - Since only limited information is provided, unanticipated data isn’t available

RRS Components

- Series of small changes:

Panic-data	panic notifiers get register contents
vlram	pseudo-device to store data in persistent memory, i.e. memory preserved over reboots
conslogger	pseudo-console for recording and diverting console output
panic_log	generates the panic report, calling a cloud of functions to provide information

vlram (variable-length RAM devices)

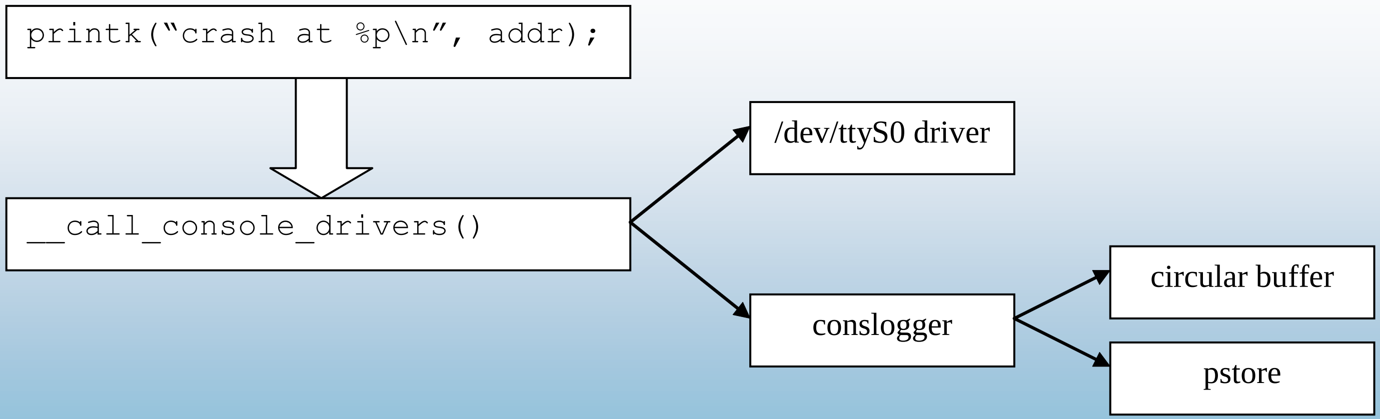
- Built on MTD:
  - Provides non-blocking reads and writes to the kernel
  - Device in /dev for user space access, /dev/mtd<n>
- Contract with bootloader and hardware to preserve contents and not overwrite area when booting
- Current implementation uses one section of DRAM to hold data, written with uncached writes
- Other implementations possible: SRAM, flash, NVM
- Also usable for providing panic log annotation

Conslogger

- Implemented as console device to obtain all console, i.e. printk(), output
- Three states:

Not logging	Output is ignored
Logging	Output stored in circular buffer for later printing
Diverting	Output passed function for storing (in vlram)

- Existing functions can be used without modification



Panic log

- Presently, this is platform-dependent
- Would be interesting to have an extensible core and standardize on some aspects of panic logs
- Needs a rich set of functions to call to report on the state of the system:
  - –The usual—registers, stack, instructions, modules
  - –State of the current process
  - –Lots more information about the system…

Additional panic log components

- IRQ logger
- /proc/meminfo information
- SoftIRQ times
- Timer times
- Biggest processes
- Current process info
  - stack dump
  - backtrace
  - /proc/<pid>/maps
  - process ancestry
  - registers
- Etc.

Status of patches

- panic-data & panic-data-<processor>: submitted
- conslogger: submitted
- vlram: prototype done, will submit soon
- panic-log: prototype done, submit soon
- other: prototypes done for the following:
  - plog-irq: IRQ history
  - plog-meminfo: /proc/meminfo data
  - plog-note: user space annotation

What’s next?

- More panic log components
  - slab top
  - concise list of current processes
  - registration of panic-log annotation, especially interesting for loadable kernel modules
  - And much more…

CISCO

Backup Material

cisco