RAS and Memory Error Reporting with perf

Robert Richter <robert.richter@linaro.org>

2nd CERN Advanced Performance Tuning workshop

November 21, 2013

Group photograph at Linaro Connect in Copenhagen
Monday 29 Oct 2012

Content

- RAS
- Hardware Error Handling
- RAS for Linux
- Perf Persistent Events
- New perf ioctls
- Perf based RAS daemon prototype

RAS - Reliability, Availability and Serviceability

- Reliability: ensure correctness of data; detect, correct (if possible) and isolate errors
- Availability: disable the malfunctioning component and continue to operate with reduced resources
- Serviceability: early detect faulty hardware and reduce time to replace it

Hardware Error Handling

Hardware error handling is important for higher levels of RAS:

- Error logging
- Error prediction
- Error recovery

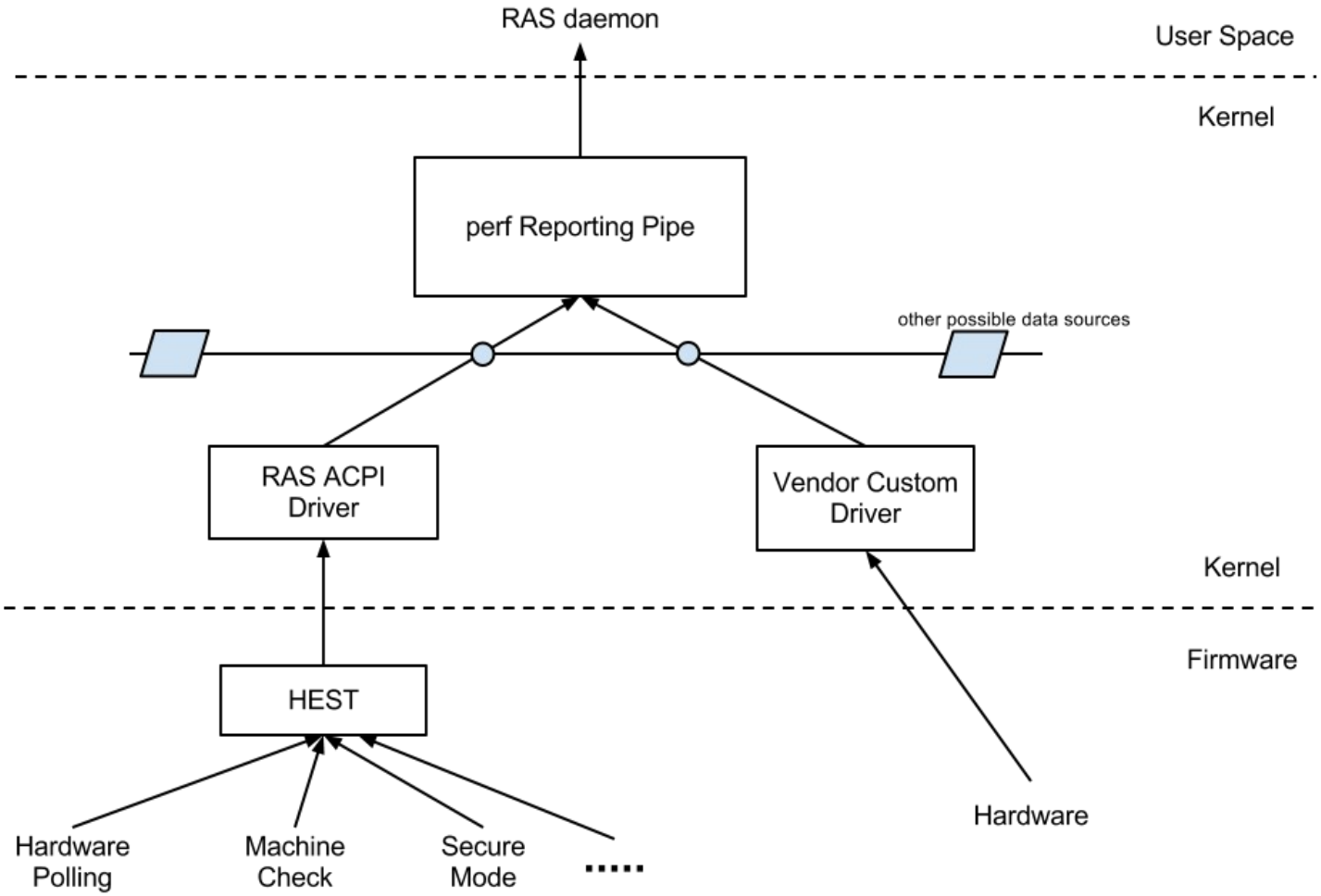
We focus on hardware error logging and reporting:

Give data center operators a tool to examine hw errors in the system for further analysis and actions (identify, disable and replace hardware components)

RAS implementation - goals

- Provide a RAS framework in the kernel to collect hardware errors from various sources and report them to userland for further processing.
- Reference implementation of a RAS daemon to enable data center operators to integrate it into their tools
- Use of perf_event_open syscall to access kernel event buffers
- Architecture independent, works on ARM and x86 (Collaborative work with Borislav Petkov from Suse)
- Upstream acceptance

RAS Framework - Technical Overview



Thanks Al, see https://wiki.linaro.org/LEG/Engineering/Kernel/ACPI/RASandACPI

RAS Components - Kernel

- Persistent perf events
  - Allows running events in the system background
- Tracepoints
  - Raw data sample, allows to transfer data structures from kernel to userland
- Hardware drivers for error detection
  - Vendor specific
  - ACPI/APEI

RAS Components - Userland

- perf tools and libraries
  - perf syscall to access event buffers
  - Extend event parser for sysfs support of persistent events
  - Move necessary perf functions into liblk
- RAS daemon
  - Reading event buffers for reporting, analysis and actions
  - Report to syslogd

Persistent events

- run standalone in the system's background
- no controlling process that holds an event's file descriptor
- always enabled
- data samples are collected in a ring buffer
- buffers are read only, sharable by multiple user for one event
- standard perf_event_open syscall to access buffers
- can be enabled by the kernel during early boot, no userland necessary to setup events
- events dynamically listed in sysfs, allows out-of-the-box event setup with perf tools

perf_event ioctls - PERF_EVENT_IOC_UNCLAIM

Create a persistent event:

- Use ioctl, event is not released when closing:

```
id = ioctl(fd, PERF_EVENT_IOC_UNCLAIM, 0);
close(fd);
```

- Event still enabled, no controlling process
- To connect to a persistent event:

```
pe.type = PERF_TYPE_PERSISTENT;
pe.config = id;
…
fd = perf_event_open(...);
```

- id known from ioctl or gathered from sysfs

perf_event ioctls - PERF_EVENT_IOC_CLAIM

Delete a persistent event:

- Re-connect to a persistent event
- “Claim” the event:

```
id = ioctl(fd, PERF_EVENT_IOC_CLAIM, 0);
/* The event is no longer persistent now */
...
close(fd);
```

- Event is released after all file descriptors to the event were closed and no process is using it anymore.

Perf tools patches

- Patch set sent that modifies perf tools to use persistent events (not yet upstream)
- Most important part: update event parser to be able to describe every event in sysfs, esp. flags (currently limited to config values of the event):

```
/sys/bus/event_source/devices/persistent/events/mce_record:persistent,config=106
/sys/bus/event_source/devices/persistent/format/persistent:attr5:23
```

- Persistent events run then out-of-the-box:

```
# perf top -e persistent/mce_record/
# perf record -e perstent/mce_record/ ...
```

Raw data defined as tracepoint

```
TRACE_EVENT(mce_record,
     TP_PROTO(struct mce *m),
     TP_ARGS(m),
     TP_STRUCT__entry(
                __field(       u64,           mcgcap          )
                __field(       u64,           mcgstatus       )
                __field(       u64,           status          )
                __field(       u64,           addr            )
                __field(       u64,           misc            )
                __field(       u64,           ip              )
                __field(       u64,           tsc             )
                __field(       u64,           walltime        )
                __field(       u32,           cpu             )
                __field(       u32,           cpuid           )
                __field(       u32,           apicid          )
                __field(       u32,           socketid        )
                __field(       u8,            cs              )
                __field(       u8,            bank            )
                __field(       u8,            cpuvendor       )
     ),
     ...
```

Persistent event patch set V3

- Posted in August, https://lkml.org/lkml/2013/8/22/306
- A couple of items has been addressed, esp. to create persistent events on-the-fly:
  - new event type PERF_TYPE_PERSISTENT introduced,
  - support for all type of events, unique event ids,
  - improvements in reference counting and locking,
  - ioctl functions are added to control persistency,
  - the sysfs implementation now uses variable list size.
  - Limitations: only system-wide events
- Reviewed by perf maintainers Ingo and PeterZ
- No further objections on the kernel part, but need support of persistent events in perf tools

RAS Prototype - Implementation

- Tracepoints collected with persistent events, recorded with perf-record and processed with perf-script
- Daemon basically doing:

```
# perf record -e persistent/mce_record/ cat /dev/null | perf script -s rasd.pl
```

- Perl script for event processing:

```
# cat rasd.pl
sub process_event
{
    my ($event, $attr, $sample, $raw_data) = @_;


    pintf("--- mce_record: ----------------\n");
    print("raw_data: ",
          ( join '',
            map { sprintf("%02x ", $_); } unpack("C*", $event) ), "\n");
     # more decoding:
     ...
}
```

RAS Prototype - Running

```
root@piledriver:~# unbuffer rasd/rasd.sh  &
[1] 2553
root@piledriver:~# echo 1 > /sys/kernel/debug/tracing/events/mce/mce_record/enable
root@piledriver:~# ./rasd/inject.sh

Message from syslogd@piledriver at Jul  5 18:38:37 ...
 kernel:[   81.182322] [Hardware Error]: MC4 Error (node 0): DRAM ECC error detected on the NB.

Message from syslogd@piledriver at Jul  5 18:38:37 ...
 kernel:[   81.201063] [Hardware Error]: Error Status: Corrected error, no action required.

Message from syslogd@piledriver at Jul  5 18:38:37 ...
 kernel:[   81.209844] [Hardware Error]: CPU:0 (15:2:0) MC4_STATUS[Over|CE|MiscV|-|AddrV|-|-|CECC]:
0xdc68c0002b080813

Message from syslogd@piledriver at Jul  5 18:38:37 ...
 kernel:[   81.223874] [Hardware Error]: MC4_ADDR: 0x000000042cd330a0

Message from syslogd@piledriver at Jul  5 18:38:37 ...
 kernel:[   81.229363] [Hardware Error]: cache level: L3/GEN, mem/io: MEM, mem-tx: RD, part-proc:
SRC (no timeout)
--- mce_record: ----------------
raw_data: 09 00 00 00 01 00 70 00 64 00 00 00 49 00 14 00 00 00 00 00 07 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 13 08 08 2b 00 c0 68 dc a0 30 d3 2c 04 00 00 00 00 00 00 01 58 02 18 c0 00 00
...
```

RAS Prototype - Running (2)

```
...
--- mce_record: ----------------
raw_data: 09 00 00 00 01 00 70 00 64 00 00 00 49 00 14 00 00 00 00 00 07 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 13 08 08 2b 00 c0 68 dc a0 30 d3 2c 04 00 00 00 00 00 00 01 58 02 18 c0 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 8d f6 d6 51 00 00 00 00 00 00 00 00 20 0f 60 00 00 00 00 00
00 00 00 00 00 04 02 00 00 00 00 00 00 00 00 00

 perf_event_type = 0x00000009
      header_misc = 0x0001
            size = 0x0070
      raw_size = 0x00000064
trace_entry_type = 0x0049
            flags = 0x14
   preempt_count = 0x00
            pid = 0x00000000
            mcgcap = 0x0000000000000107
      mcgstatus = 0x0000000000000000
            status = 0xdc68c0002b080813
            addr = 0x000000042cd330a0
            misc = 0xc018025801000000
            ip = 0x0000000000000000
            tsc = 0x0000000000000000
      walltime = 0x0000000051d6f68d
            cpu = 0x00000000
            cpuid = 0x00600f20
...
```

RAS Prototype - Running (3)

```
…
      walltime = 0x0000000051d6f68d
            cpu = 0x00000000
            cpuid = 0x00600f20
            apicid = 0x00000000
      socketid = 0x00000000
            cs = 0x00
            bank = 0x04
      cpuvendor = 0x02
^C
root@piledriver:~# cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
# entries-in-buffer/entries-written: 1/1   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                             | / _---=> hardirq/softirq
#                             || / _--=> preempt-depth
#                             ||| /      delay
#           TASK-PID   CPU#   |||| TIMESTAMP  FUNCTION
#              | |       |    ||||         |         |
           <idle>-0   [000] .Ns. 81.182316: mce_record: CPU: 0, MCGc/s: 107/0, MC4:
dc68c0002b080813, ADDR/MISC: 000000042cd330a0/c018025801000000, RIP: 00:<0000000000000000>, TSC: 0,
PROCESSOR: 2:600f20, TIME: 1373042317, SOCKET: 0, APIC: 0
```

RAS - Next Steps

- Enable RAS framework on ARM
- Integrate ACPI/APEI
- Split perf tool code into liblk
- Implement RAS daemon

RAS and Memory Error Reporting with perf

Questions?