In-kernel memory-mapped I/O tracing

Home page and links to optional user space tools:

https://nouveau.freedesktop.org/wiki/MmioTrace

MMIO tracing was originally developed by Intel around 2003 for their Fault Injection Test Harness. In Dec 2006 - Jan 2007, using the code from Intel, Jeff Muizelaar created a tool for tracing MMIO accesses with the Nouveau project in mind. Since then many people have contributed.

Mmiotrace was built for reverse engineering any memory-mapped IO device with the Nouveau project as the first real user. Only x86 and x86_64 architectures are supported.

Out-of-tree mmiotrace was originally modified for mainline inclusion and ftrace framework by Pekka Paalanen <pq@iki.fi>.

Preparation

Mmiotrace feature is compiled in by the CONFIG_MMIOTRACE option. Tracing is disabled by default, so it is safe to have this set to yes. SMP systems are supported, but tracing is unreliable and may miss events if more than one CPU is on-line, therefore mmiotrace takes all but one CPU off-line during run-time activation. You can re-enable CPUs by hand, but you have been warned, there is no way to automatically detect if you are losing events due to CPUs racing.

Usage Quick Reference

```
$ mount -t debugfs debugfs /sys/kernel/debug
$ echo mmiotrace > /sys/kernel/debug/tracing/current_tracer
$ cat /sys/kernel/debug/tracing/trace_pipe > mydump.txt &
Start X or whatever.
$ echo "X is up" > /sys/kernel/debug/tracing/trace_marker
$ echo nop > /sys/kernel/debug/tracing/current_tracer
Check for lost events.
```

Usage

Make sure debugfs is mounted to /sys/kernel/debug. If not (requires root privileges):

```
$ mount -t debugfs debugfs /sys/kernel/debug
```

Check that the driver you are about to trace is not loaded.

Activate mmiotrace (requires root privileges):

```
$ echo mmiotrace > /sys/kernel/debug/tracing/current_tracer
```

Start storing the trace:

```
$ cat /sys/kernel/debug/tracing/trace_pipe > mydump.txt &
```

The 'cat' process should stay running (sleeping) in the background.

Load the driver you want to trace and use it. Mmiotrace will only catch MMIO accesses to areas that are ioremapped while mmiotrace is active.

During tracing you can place comments (markers) into the trace by $ echo "X is up" > /sys/kernel/debug/tracing/trace_marker This makes it easier to see which part of the (huge) trace corresponds to which action. It is recommended to place descriptive markers about what you do.

Shut down mmiotrace (requires root privileges):

```
$ echo nop > /sys/kernel/debug/tracing/current_tracer
```

The 'cat' process exits. If it does not, kill it by issuing 'fg' command and pressing ctrl+c.

Check that mmiotrace did not lose events due to a buffer filling up. Either:

```
$ grep -i lost mydump.txt
```

which tells you exactly how many events were lost, or use:

```
$ dmesg
```

to view your kernel log and look for "mmiotrace has lost events" warning. If events were lost, the trace is incomplete. You should enlarge the buffers and try again. Buffers are enlarged by first seeing how large the current buffers are:

```
$ cat /sys/kernel/debug/tracing/buffer_size_kb
```

gives you a number. Approximately double this number and write it back, for instance:

```
$ echo 128000 > /sys/kernel/debug/tracing/buffer_size_kb
```

Then start again from the top.

If you are doing a trace for a driver project, e.g. Nouveau, you should also do the following before sending your results:

```
$ lspci -vvv > lspci.txt
$ dmesg > dmesg.txt
$ tar zcf pciid-nick-mmiotrace.tar.gz mydump.txt lspci.txt dmesg.txt
```

and then send the .tar.gz file. The trace compresses considerably. Replace "pciid" and "nick" with the PCI ID or model name of your piece of hardware under investigation and your nickname.

How Mmiotrace Works

Access to hardware IO-memory is gained by mapping addresses from PCI bus by calling one of the ioremap_*() functions. Mmiotrace is hooked into the __ioremap() function and gets called whenever a mapping is created. Mapping is an event that is recorded into the trace log. Note that ISA range mappings are not caught, since the mapping always exists and is returned directly.

MMIO accesses are recorded via page faults. Just before __ioremap() returns, the mapped pages are marked as not present. Any access to the pages causes a fault. The page fault handler calls mmiotrace to handle the fault. Mmiotrace marks the page present, sets TF flag to achieve single stepping and exits the fault handler. The instruction that faulted is executed and debug trap is entered. Here mmiotrace again marks the page as not present. The instruction is decoded to get the type of operation (read/write), data width and the value read or written. These are stored to the trace log.

Setting the page present in the page fault handler has a race condition on SMP machines. During the single stepping other CPUs may run freely on that page and events can be missed without a notice. Re-enabling other CPUs during tracing is discouraged.

Trace Log Format

The raw log is text and easily filtered with e.g. grep and awk. One record is one line in the log. A record starts with a keyword, followed by keyword- dependent arguments. Arguments are separated by a space, or continue until the end of line. The format for version 20070824 is as follows:

Explanation Keyword Space-separated arguments

read event R width, timestamp, map id, physical, value, PC, PID write event W width, timestamp, map id, physical, value, PC, PID ioremap event MAP timestamp, map id, physical, virtual, length, PC, PID iounmap event UNMAP timestamp, map id, PC, PID marker MARK timestamp, text version VERSION the string "20070824" info for reader LSPCI one line from lspci -v PCI address map PCIDEV space-separated /proc/bus/pci/devices data unk. opcode UNKNOWN timestamp, map id, physical, data, PC, PID

Timestamp is in seconds with decimals. Physical is a PCI bus address, virtual is a kernel virtual address. Width is the data width in bytes and value is the data value. Map id is an arbitrary id number identifying the mapping that was used in an operation. PC is the program counter and PID is process id. PC is zero if it is not recorded. PID is always zero as tracing MMIO accesses originating in user space memory is not yet supported.

For instance, the following awk filter will pass all 32-bit writes that target physical addresses in the range [0xfb73ce40, 0xfb800000]

```
$ awk '/W 4 / { adr=strtonum($5); if (adr >= 0xfb73ce40 &&
adr < 0xfb800000) print; }'
```

Tools for Developers

The user space tools include utilities for:

- replacing numeric addresses and values with hardware register names
- replaying MMIO logs, i.e., re-executing the recorded writes