Multiprocessor MMIO Tracing via Memory Protection and a Shadow Page Table

Myoungjae Kim[1], Hyunmin Yoon[2], Minkwan Choi[1], Shakaiba Majeed[1], and Minsoo Ryu[1*]
[1]Department of Computer Science and Engineering, Hanyang University, Seoul, Korea
[2]Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea
{mjkim, hmyoon, mkchoi, shakaiba}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

Abstract – *Memory-mapped I/O (MMIO) tracing provides an effective means for analyzing and debugging I/O related functions since it allows us to observe and track the interplay between processors and I/O devices [1]. However, existing MMIO tracing techniques have a serious drawback in multicore systems. Current MMIO techniques commonly use a memory protection mechanism to detect access to an MMIO address area under consideration. Unfortunately, this approach may miss some I/O events and even lead to a data race condition due to inappropriate management of concurrent accesses to the MMIO address area. In this paper, we describe a novel MMIO tracing approach introducing the notion of shadow page table. We use a shadow page table to allow only one processor to have access to a MMIO address area while forbidding other processors' access to the same MMIO address area. We show how the shadow page table approach can be efficiently implemented on a multiprocessor platform with dual core ARM Cortex A15 CPU.*

Keywords: Memory Mapped I/O (MMIO) Trace, Memory Protection, Page Fault, Shadow Page Table.

1 Introduction

Memory-mapped I/O (MMIO) tracing provides an effective means for analyzing and debugging I/O related functions since it allows us to observe and track the interplay between processors and I/O devices. For example, to analyze and debug failures in device drivers, developers must be able to find out what data is sent to or received from the device. MMIO tracing can collect detailed information about I/O operations conducted between a processor and I/O devices, thus enabling us to track down the source of failures.

However, existing MMIO tracing techniques have a serious drawback in multicore systems. Current MMIO tracing techniques commonly use a system-wide address translation table, i.e. page table in processors with paging support, to set the MMIO address area under consideration as invalid and rely on memory access exceptions to detect any processor's access to the protected MMIO address area. When an exception is generated by a read/write instruction, a specially designed exception handler collects information about the I/O access, enables access permission for the MMIO address area, re-executes the faulting memory access instruction, and sets the access permission back to invalid. Unfortunately, in multicore hardware, this may lead to missing some I/O events and even a data race condition since other processors can make writes simultaneously to the same address area during the time interval where the access to MMIO address area is enabled.

In this paper, we present a novel MMIO tracing method introducing the notion of shadow page table. When a page fault occurs on a certain processor, we replace the page table seen by the exception handling processor with a shadow page table, while leaving other processors referencing the original page table. The shadow page allows only the exception handling processor to access the MMIO address area, but other processors' access to the MMIO area is prohibited through the original page table. Therefore, this approach allows us to avoid the problem of missing I/O events and race conditions. We describe how the shadow page table approach can be efficiently implemented on a multiprocessor platform with dual core ARM Cortex A15 CPU.

This paper is organized as follows. Section 2 describes existing MMIO tracing techniques. Section 3 presents our shadow page table approach and Section 4 concludes this paper.

2 Background of MMIO Tracing

2.1 Memory-mapped I/O (MMIO)

MMIO requires a section of memory to allow a processor to communicate with I/O controllers. A processor with MMIO support reserves some part of its address space for a special I/O address range where I/O controllers' registers are mapped to specific addresses in the designated I/O address range. Programs can access I/O registers through memory access instructions such as load and store, which is no different from read/write access to normal memory addresses [3].

MMIO tracing can be efficiently implemented using a page table. A page table contains the mapping between virtual addresses and physical addresses and some additional information associated with each page table entry. One important piece of information is the access permission for

each page. By manipulating the access permission for each MMIO page, we can allow or prohibit the processor's access to specific MMIO pages. MMIO tracing initially disables access permission for MMIO pages using the page table. Whenever a processor attempts to access a protected MMIO page, a page fault exception occurs. A special page fault handler then collects information about the I/O access, enables the access permission for the MMIO page, re-executes the faulting memory access instruction, and re-disables the access permission.

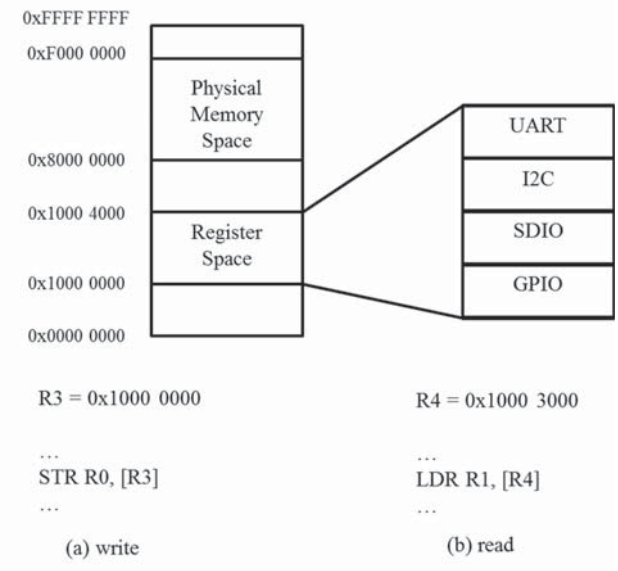

Figure 1. Address space of a Processor using MMIO.

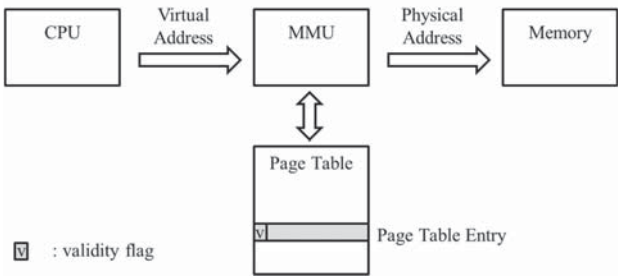

Figure 2. Paging and translation scheme.

2.2 MMIO Tracing in Linux

The Linux MMIO tracing tool uses a validity attribute associated with each page table entry to force page fault to occur when a processor accesses a memory mapped I/O region even if the region exists in a valid page [5]. The tool records the MMIO accesses in the following way: First, the MMIO pages are marked as invalid. When a fault occurs due to an access to these pages, the page fault handler emulates the faulting instruction by changing the attribute of the page as valid and starts logging the events. After the emulation and logging the page fault handler again marks the page as invalid. Finally, the interrupted kernel code takes control again and executes the next instruction to the faulting instruction.

While the page fault handler is emulating the faulting instruction, the other processors can freely access the page containing the data which the faulting instruction wanted to access because that page is marked valid during this interval. In such situation, other processor's access does not create a page fault which leads to event missing without notice.

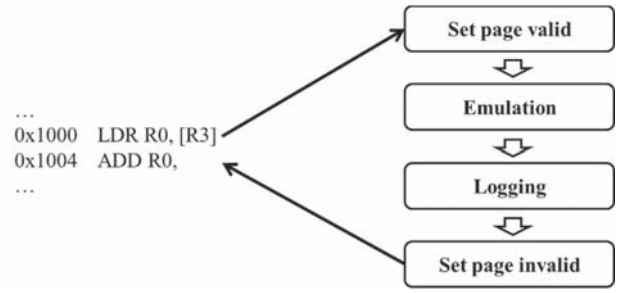

Figure 3. Tracing control flow.

3 MMIO Tracing with a Shadow Page Table

As mentioned earlier, existing MMIO tracing techniques based on a memory protection mechanism may fail to capture some concurrent I/O events on multiprocessors. The problem is that other processors can make references to the same MMIO address area during the interval the memory access is allowed. Those accesses cannot be detected as they do not trigger page fault exceptions and may even lead to data race conditions.

A plausible solution is freezing other processors during the page fault handling. When a page fault happens, we may stop other processors' execution by sending a special inter-processor interrupt (IPI) to other processors. This would prevent other processors from accessing the MMIO address area. However, sending and receiving IPIs also requires access to the interrupt controller's MMIO addresses, which would entail the same problem.

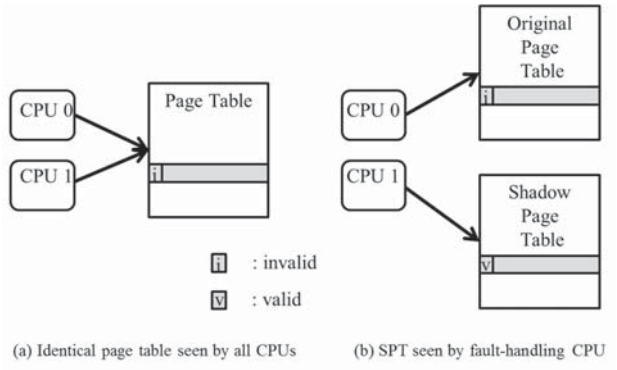

Figure 4. Shadow Page Table.

In order to address the above problem, we propose the use of a shadow page table (SPT). When a page fault occurs, a shadow page table replaces the kernel's original page table used by the fault handling processor. The use of shadow page table allows us to enable the access permission of the fault handling processor while other processors' memory access is prohibited by the original kernel's page table. Therefore, this approach can overcome the problem of missing I/O events and race conditions.

The shadow page table can be efficiently implemented in many operating systems that support paging-based memory management. We replicate the original kernel's page table and modify the access rights to the MMIO address areas in the replicated shadow page table to enable access permission. When a page fault occurs, we change the page table base register of the processor so that it can refer to the shadow page table during the page fault handling. Since other processors still refer to the original page table, they are not allowed to make access to the MMIO address areas. Once logging MMIO I/O access information is done, we change the page table base register to point to the original page table. Afterwards, all the processors use the original page table. There is a possibility that two more processors try to write access on a same MMIO address almost at the same time. It also leads to data race condition as two processors re-execute the faulting memory access instructions. To prevent this problem, we need to protect fault handling as a critical section with a synchronization method such as spin lock.

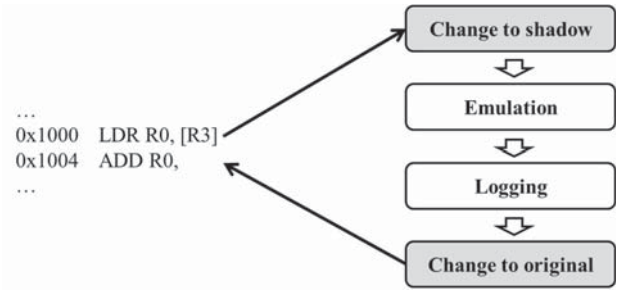

Figure 5. Tracing control flow with SPT.

4 Conclusion

In this paper, we have presented a novel MMIO tracing method introducing the notion of shadow page table. Letting a processor refer to shadow page table while it conducts MMIO tracing, we can solve a problem of missing another MMIO event by other processors as well as data race condition under multiprocessor platform.

5 Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), and partly by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (NRF-2011-0015997), and partly the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) (IITP-2015-H8601-15-1005) supervised by the IITP(Institute for Information & communications Technology Promotion).

6 References

[1] Wikipedia, "Memory-mapped I/O," [Online]. Available: http://en.wikipedia.org/wiki/Memory-mapped_I/O

[2] A. Kadav and M. M. Swift, "Understanding modern device drivers," *ACM SIGARCH Computer Architecture News*, vol. 40, pp. 87-98, 2012.

[3] D. P. Bovet and M. Cesati, *Understanding the Linux kernel*: " O'Reilly Media, Inc.", 2005

[4] Wikipedia, "Virtual memory," [Online]. Available: http://en.wikipedia.org/wiki/Virtual_memory

[5] LWN, "Tracing memory-mapped I/O operations," [Online]. Available: https://lwn.net/Articles/270939/