Deploying LTTng on Exotic Embedded Architectures

Mathieu Desnoyers
École Polytechnique de Montréal
mathieu.desnoyers@polymtl.ca

Michel R. Dagenais
École Polytechnique de Montréal
michel.dagenais@polymtl.ca

Abstract

This paper presents the considerations that comes into play when porting the LTTng tracer to a new architecture. Given the aleady portable design of the tracer, very targeted additions can enable tracing on a great variety of architectures. The main challenge is to understand the architecture time-base in enough depth to integrate it to LTTng by creating a lock-less fast and fine-grained trace clock. The ARM OMAP3 LTTng port will be used as an example.

1 Introduction

Gathering an execution trace at the operating system level has been an important part of the embedded system development process, as illustrates the case of the Mars Pathfinder[5] priority inversion, solved by gathering an execution trace.

In the Linux environment, such system-wide facility is just becoming accepted in the mainline kernel, but with the kernel developer audience as main target. The portability issues and lack of user-oriented tools to manipulate the information may be indications of two points where the current state of Linux kernel tracing is not up to the tools embedded systems developers are used to. For instance, WindRiver is shipping LTTng with their Linux distribution to provide similar features as in VxWorks. Montavista also integrates LTTng to their Carrier Grade Linux distribution.

LTTng currently supports the following architectures :

- X86 32/64
- MIPS
- PowerPC 32/64
- ARM[1]
- S390 [2]
- Sparc 32/64[3]
- SH64[4]

This paper presents the key abstractions required to port LTTng to a new architecture, mainly describing what considerations must be taken into account when designing a trace clock time source suitable for tracing, meeting the platform-specific constraints.

[1] Limited time-stamping precision
[2] Partial instrumentation
[3] Partial instrumentation
[4] Partial instrumentation

2 State of the Art

Looking at some of the most widely used open-source tracing alternatives, we can see that Dtrace[3], Ftrace (as of Linux kernel 2.6.29) and SystemTAP [4] are three of the solutions that are the closest to LTTng.

Dtrace does not require the same level of portability, given Solaris only targets Intel x86 and Sparc architectures. Therefore, this tracer assumes access to a time-stamp counter register.

Ftrace uses the scheduler time-source due to its low performance impact. However, on 32-bits architecture, this time-source is using 64-bits variables without using the proper sequence lock synchronization primitives. While having a scheduler doing a wrong decision once in a while is acceptable, having incorrect timing information in a trace can be very confusing.

SystemTAP is using the time source primitives provided by the Linux kernel. Those are taking a sequence lock to protect non-atomic data structure accesses. This implies that instrumentation coverage cannot include non-maskable interrupt handlers `NMIs`, because a `NMI` taking a sequence read lock nested over a sequence write lock would deadlock.

3 Porting LTTng to a new architecture

In the case of LTTng, even though it uses very efficient binary buffers to extract the data out of kernel-space, all its design deals with type size and endianness issues by making sure the information written in the trace is self-described by a *meta-data* information channel.

We will now see what work still has to be done when porting LTTng to a new architecture.

3.1 Instrumentation

First, the instrumentation must be expanded to include some architecture-specific events.

System call instrumentation has to be added at system call entry and exit *syscall_trace*. LTTng already adds a thread flag to the low-level kernel assembly to call into the standard system call tracing callback of each architecture supported by the Linux kernel.

Given that LTTng instrumentation covers architecture-agnostic sites for most of its event sources, only few supplementary architecture-specific tracepoints are required :

- *kernel_thread_create*
- *syscall_trace*
- *ipc_call*[5]
- *trap_entry, trap_exit*
- *page_fault_entry*, *page_fault_exit*

In some cases (x86 for instance), a few interrupt handlers do not use the architecture-agnostic code to connect to their interrupt vector. In those cases, new *irq_entry* and *irq_exit* events are needed in the architecture code.

3.2 Time Source

3.2.1 Architectural Variety

The time sources available on different architectures vary greatly. On x86-based architectures, the `tsc` register[2][6] can usually be used as a fast and fine-grained time source. The

[5] `IPC` : Inter-Process Communication

PowerPC has the `tb` register which allows all cores to read a common time source through a register read.

However, embedded architectures like ARM and MIPS usually either depend on an external memory-mapped I/O timer to provide a time-source, or on a 32-bits only register read.

One of the main problems encountered with tracing clock source design is the different limitations founds in various architectures. For instance, MIPS and ARM only provide 32-bits clock sources, either through a local register or through external memory mapped I/O. These counters overflow too frequently to be usable per se without being extended to 64-bits.

Some architectures provide fast register-based time counters that are not synchronized across cores. In those cases, more evolved algorithms must be deployed to make the time base appear reasonably synchronized across cores.

Yet another problem faced is that some architectures (e.g. ARM OMAP3) stops the CPU time base register counter in sleep modes. The same appears on various AMD and Intel processors which stop the `tsc` register during idle.

3.2.2 Characteristics Required

The main concern, when porting LTTng to a different architecture, is to create a trace clock suited to the architecture restrictions which offers the following characteristics.

LTTng relies on a 64-bits monotonic time base available system-wide. 32-bits would be too few, given it would overflow every 4.29 seconds at 1 GHz. A channel only recording rare events would not allow to detect longer time deltas accurately.

The tracer also needs the timer presented to the tracer to appear as being reasonably synchronized across cores. At least, a maximum bound on the time precision error should be ensured.

The time-base needs to be fast, given it has to be read for each event record. This can easily drag system's resources. It also needs to be scalable if the architecture aims at supporting *SMP*. Any data structure used to keep track of time-base state should be designed so it can be read from any context. Therefore, locking must be chosen with great care.

3.2.3 Design Considerations

Given those requirements, some design choices become clear. First, whenever a cycles counter register is available on the architecture, it should be used as a per-event time source.

External memory-mapped I/O timers should only be used as a time source when strictly required, because they are typically much slower than the cycle counter register, and do not scale well. The `HPET`[6] has been measured to add respectively a 10%, 11% and 19% performance impact to LTTng on 1, 2 and 8 cores, while the performance impact is constant with a scalable time-source like the cycle counter. However, those external timer counters can be useful to resynchronize the cycle counter when coming back from sleep or idle, if these events do not occur too frequently.

Regarding locking, the sequence lock (*seqlock*) should be avoided, because it would cause deadlocks if used in a NMI handler. It would also deadlock if instrumentation is added to a code path protected by the write sequence lock.

The best solution is therefore to use RCU-like algorithms to manage data updates. It lets any

[6] `HPET` : High-Precision Event Timer[1]

reader context successfully read a valid copy of the data.

Extension of 32-bits to 64-bits timestamps is provided by the LTTng *trace-clock-32-to-64* module, which lets any kernel context read the 64-bits time-base atomically using a RCU-like algorithm.

4 ARM OMAP3 Example

Those general guidelines being drawn, this section shows how they can be applied to a real-life scenario. This involves porting LTTng to the ARM OMAP3 architecture.

The ARM OMAP3 has a 31-bits (usable) cycle counter register called *ccnt*. A hardware bug prevents using the 32nd bit because of a race between the CP14 and CP15 coprocessor register accesses and the *ccnt* register overflow. The solution is to periodically clear the high order bit to make sure overflow never happens. The *trace-clock-32-to-64* code is actually flexible enough to have the number of lower "hardware" bits to expect specified at compile-time. Therefore, by overriding the number of hardware bits to 31, it supports the buggy cycle counter register without problem.

Sleep support is provided by allowing to resynchronize the *ccnt* register and the *trace-clock-32-to-64* data structures by reading the 32k timer upon return from sleep. This timer has the particularity of not stopping even in deep sleep mode. Given it runs at a lower frequency, the overall precision of timestamps will be lower, but it should be enough for general purpose tracing needs (30 μs granularity). This should therefore be taken into account in the time-base imprecision when analysing the traces taken from such architecture, especially if it involves a SMP system.

5 Variable Frequency Support Discussion

Architectures like the ARM OMAP3 and some AMD and Intel processors base the time-stamp counter register on the CPU's frequency, which may vary depending on various conditions. The kernel can ask the frequency to be continuously adapted to match the workload, temperature sensors can slow down the CPU frequency to make sure it does not overheat, etc. This frequency modulation is usually done per-core, which means that the cycle counter register will be non-synchronized across cores. Newer Intel and AMD architectures fix this issue by letting the *tsc* register run at a lower, synchronized, frequency.

However, those architectures with non-synchronized cycle counter register still have to be supported. If we are lucky, the OS is informed of the frequency change and calls a notifier chain. In the opposite case, e.g. some AMD CPUs, the speed is throttled by the Southbridge without notifying the operating system by an interrupt.

Even if we are lucky, the delay between the actual frequency change and the delivery of the frequency change notification would generate an imprecision between the real reference time and the CPU time. For instance, a CPU going from 1 GHz to 2 GHz, assuming approximately 5000 cycles between interrupt reception and data structure updates, will count 5000 cycles which should be considered as 2500 cycles in reference time. Therefore, 2500 cycles at 1 GHz bring a 2.5μs offset. Given that the delay between interrupt and update is highly variable due to cache effect, higher priority interrupts and interrupt disabled regions, the error will accumulate.

Even worse, interrupt disabled sections in the Linux kernel may last for as long as 15μs.

Given the cumulative nature of this error, it quickly becomes problematic for SMP synchronization.

Another alternative would be to use a HPET-like time-source, but it involves a slow memory-mapped I/O access and it does not scale when the number of CPUs increase. Otherwise, using a compare-and-swap instruction to make sure the time-base never goes backward could help, but the cache-line bouncing involved makes it completely non-scalable.

My proposal, which might have some value if proven workable, would be to use a periodical memory-mapped external clock source resynchronization. This would make sure the error does not accumulate. When a frequency change is about to be requested by the operating system, the time-stamp should be saved, all this with interrupts disabled. This would make sure the time-stamp value matches as closely as possible the moment at which the frequency has been changed. The CPU frequency change notifier should then be used to deal with cases where the OS is not requesting the frequency change. If this happens rarely enough, it would be valuable to perform an external clock-source based resynchronization at that point.

6 Conclusion

Porting LTTng to new architectures, as this paper detailed, involves mainly supporting the architecture-specific hardware time sources, either cycle counter registers or external memory-mapped I/O clock sources. The mix of constraints being : speed, scalability, reentrancy and fine granularity in a SMP environment makes this problem relatively hard to tackle. LTTng includes helpers such as *trace-clock-32-to-64* to make the port task easier. In the event that an architecture needs more logic to deal with non-synchronized cycle counters or with sleep modes, great care must be taken to use data structures that will let the time-base be read in any execution context.

Most of LTTng's instrumentation is already architecture-agnostic, so only a small subset of instrumentation needs to be added to the architecture source tree. As far as portability of data structures used in LTTng, all the tracer is portable between various platforms and usable between different endianness.

References

[1] High precision event timers (hpet) specification. http://www.intel.com/technology/architecture/hpetspec.htm, October 2004.

[2] Intel® pentium® m processor on 90 nm process with 2-mb l2 cache specification update and intel® processor a100 and a110 on 90 nm process with 512-kb l2 cache. http://www.intel.com/design/mobile/specupdt/302209.htm, November 2007.

[3] Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. Dynamic instrumentation of production systems. In *USENIX '04*, 2004.

[4] Vara Prasad, William Cohen, Frank Ch. Eigler, Martin Hunt, Jim Keniston, and Brad Chen. Locating system problems using dynamic instrumentation. In *OLS (Ottawa Linux Symposium) 2005*, 2005.

[5] G. E. Reeves. What really happened on mars? (technical report). http://hdl.handle.net/2014/19020, February 1998.

[6] AMD Fellow Rich Brunner. Tsc and power management events on amd processors. http://lkml.org/lkml/2005/11/4/173, November 2005.