

> Linux Trace Toolkit Next Generation (LTTng)

Tracing across execution layers, from the
Hypervisor to user-space.

Mathieu Desnoyers
Ottawa Linux Symposium 2008

> Mathieu Desnoyers

- Maintainer of
 - LTTng
 - LTTV
- Author of the Linux Kernel Markers
- Completing a Ph.D. in computer engineering at École Polytechnique de Montréal

> Plan

- Tracing infrastructure
- Mainlining status and plans
- Immediate Values
- Efficient user-space tracing
- Hypervisor tracing
- Conclusion

> Tracing Infrastructure

- Instrumentation
 - Kernel
 - Userspace
- Probes
 - Specialized
 - Generic
- Data extraction

> Mainlining status (1)

- Currently in mainline
 - Data extraction
 - Per CPU atomic operations
 - Instrumentation
 - Linux Kernel Markers
 - Enhanced code patching support

> Mainlining status (2)

- In preliminary trees
 - Tracepoints
 - Immediate Values
 - Text Edit Lock

> Short-term submission plan (1)

- In LTTng patchset
 - Instrumentation
 - LTTng tracepoints
 - Used by LTTng, SystemTAP and specialized probes.
 - Port specific sets of tracepoints along with their current users
 - ftrace, KVM trace, blktrace.

> Short-term submission plan (2)

- In LTTng patchset
 - Data extraction
 - LTTng time-stamping
 - LTTng trace management
 - LTTng data relay
 - LTTng marker control

> Medium-term submission

- In LTTng patchset
 - Instrumentation
 - Userspace tracing interface
 - LTTng statedump

> Linux Kernel Markers

Marker example :

```
trace_mark(kernel_sched_schedule,  
            "prev_pid %d next_pid %d prev_state %ld",  
            prev->pid, next->pid, prev->state);
```

- Unique name
- Field names
- Typing information
 - Meant to be exported to user-space
 - Does not contain specific pointer typing information

> Linux Kernel Markers

- What is really produced
 - Jump over stack setup and function call
 - Comparatively, Dtrace nops out the function call, but leaves the stack setup.
 - Stack setup and function call are placed in an unlikely branch at the end of the function
 - Dtrace puts it in cache-hot instructions.
- Can connect multiple probes dynamically
- Type-checking done by format-string comparison

> Tracepoints

include/trace/sched.h :

```
DEFINE_TRACE(sched_switch,  
             TPPROTO(struct rq *rq, struct task_struct *prev,  
                    struct task_struct *next),  
             TPARGS(rq, prev, next));
```

kernel/sched.c :

```
#include <trace/sched.h>
```

```
    trace_sched_switch(rq, prev, next);
```

kernel/trace/sched-trace.c :

```
#include <trace/sched.h>
```

```
void my_sched_switch_tracer(struct rq *rq,  
    struct task_struct *prev, struct task_struct *next);
```

```
register_trace_sched_switch(my_sched_switch_tracer);
```

> Tracepoints

- Can connect multiple probes dynamically
- Type-checking done by the compiler
- Permits to express complex types, such as pointers to specific structures

> Immediate Values : conditional branch

Standard branch

```
b2c8: 80 3d 00 00 00 00 00  cmpb  $0x0,0x0(%rip)
                                # b2cf <wake_up_new_task+0x9f>
b2cf: 75 47                    jne   b318 <wake_up_new_task+0xe8>
```

Short jump depending on an immediate value (movb, off)

```
b2c8: b0 00                    mov   $0x0,%al
b2ca: 84 c0                    test  %al,%al
b2cc: 75 42                    jne   b310 <wake_up_new_task+0xe0>
```

Short jump depending on an immediate value (movb, on)

```
b2c8: b0 01                  mov   $0x1,%al
b2ca: 84 c0                    test  %al,%al
b2cc: 75 42                    jne   b310 <wake_up_new_task+0xe0>
```

> Conditional branches

- Eliminates the d-cache hit
- Removes 3 bytes of cache-hot instructions (6 bytes instead of 9)
- Pollutes the BPB

> Immediate Values static jump patching

Short jump depending on an immediate value (movl)

```
b2c8:    b8 00 00 00 00    mov    $0x0,%eax
b2cd:    85 c0             test   %eax,%eax
b2cf:    75 47            jne    b318 <wake_up_new_task+0xe8>
```

After static jump patching (disabled)

```
b2c8:    c9 04 00 00 00 jmp    b2d1
b2cd:    85 c0             test   %eax,%eax
b2cf:    75 47            jne    b318 <wake_up_new_task+0xe8>
b2d1:    ....
```

After static jump patching (enabled)

```
b2c8:    c9 4B 00 00 00 jmp    b318
b2cd:    85 c0             test   %eax,%eax
b2cf:    75 47            jne    b318 <wake_up_new_task+0xe8>
b2d1:    ....
```

> Immediate Values code pattern matching

- GCC optimizations are not our friend
 - Detect the code pattern following the patch site
 - Register liveness analysis
 - Add an asm statement which clobbers the output register in both branches
 - Inline asm puts the current IP in a special section
 - Make sure the compiler cannot save the variable in memory
 - Fallback on conditional branch if the expected pattern is not found

> Immediate Values (speedup)

Speedups with uncached memory accesses

x86 Pentium 4, 3.0GHz, Linux 2.6.25-rc7	Added cycles (cached)	Added cycles (uncached)
Optimized marker	0.002	0.07
Normal marker	0.004	154.7
Stack setup + (1+4 bytes) NOPs (6 local var.)	0.04	0.6
Stack setup + (1+4 bytes) NOPs (1 pointer read, 5 local var.)	0.03	222.8

Table 1: Comparison of markers and disabled function impact on x86_32

AMD64, 2.0GHz, Linux 2.6.25-rc7	Added cycles (cached)	Added cycles (uncached)
Optimized marker	-1.2	0.2
Normal marker	-0.3	41.8
Stack setup + (1+4 bytes) NOPs (6 local var.)	-0.5	0.01
Stack setup + (1+4 bytes) NOPs (1 pointer read, 5 local var.)	2.7	51.8

Table 2: Comparison of markers and disabled function impact on x86_64

> Immediate Values

- Downside of the static jump patching approach
 - Its complexity
- Wish list
 - GCC support for static branch patching
 - Would eliminate requirement for code pattern matching

> Efficient User-space Tracing

- Static instrumentation
 - Markers/Tracepoints
 - port of the kernel instrumentation mechanisms
 - Early boot : tracing init
- Data extraction
 - Through the kernel
 - In user-space
 - Extract traces of killed applications and/or crashed kernels

> User-space Static Instrumentation

- Add a `sys_marker` system call
 - Registers/Unregisters markers from the kernel
 - Kernel activates the markers by writing to user-space memory
 - At registration and asynchronously
- Support executables and shared libraries
- Declares a new markers (tracepoints) section
 - Modification of the linker scripts

> Inefficient User-space Tracing

- Add a `sys_trace` system call
- Called each time an event must be recorded
- Easy to gather crash information
- Dealing with various field types difficult
 - Cannot pass variable args. to the kernel
 - Currently encode in string
- Slow

> Efficient User-space Tracing

- Buffers mapped in user-space
- Protect buffer data across processes
- Ideally per-cpu, or per-thread
- Cannot disable preemption
 - Cannot use per-cpu atomic operations
- Cannot use RCU (no multiple traces)
- Can block (good!) : smaller buffers ok
- Allow tracing of signal handlers

> Hypervisor Tracing

- KVM tracing
 - Proposed port of KVM instrumentation to tracepoints
 - Creation of LTTng probe set
 - Tracepoints -> Markers
- Xen tracing
 - Unmaintained LTTng Xen hypervisor tracer
 - Own set of hypervisor buffers exported to user-space through an hypercall.

> Conclusion

- We can statically instrument the Linux kernel with a very small footprint
- Need
 - GCC support for static branch patching
 - Work on efficient user-space data extraction

> Questions ?

- Project website : <http://ltd.polymtl.ca>