Dynamic Event Tracing in Linux Kernel

April 15 2010
4th Linux Foundation Collaboration Summit

Masami Hiramatsu
Principal Software Engineer
<masami.hiramatsu@hds.com>
Who am I?

- **Company**
  - Hitachi Data Systems
  - Works at Red Hat office as an on-site engineer

- **Linux Kernel**
  - Kprobes related matters maintainer

- **Systemtap**
  - Some enterprise/performance enhancements
<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Introduction</td>
</tr>
<tr>
<td>• Trace Events</td>
</tr>
<tr>
<td>• Kprobe-tracer</td>
</tr>
<tr>
<td>• Usage</td>
</tr>
<tr>
<td>• Usability issue</td>
</tr>
<tr>
<td>• Perf probe</td>
</tr>
<tr>
<td>• Usage</td>
</tr>
<tr>
<td>• Options</td>
</tr>
<tr>
<td>• Tips</td>
</tr>
<tr>
<td>• Kprobe Jump optimization</td>
</tr>
<tr>
<td>• Conclusion</td>
</tr>
</tbody>
</table>
There are many tracing facilities in kernel today

- Ftrace
- Tracepoints
- perf_events

These provide fixed tracing points or hardware events

Dynamic event tracing has been introduced in 2.6.33

- A few people knows how to use it.
- This slide will explain it.
Tracing Events

- **Fixed Events**
  - Tracepoints - Static event tracing
  - Mcount - Function entry(exit) tracing

- **Hardware Events**
  - Performance counters - HW event tracing
  - HW Breakpoint - HW memory access tracing

- **Dynamic Events**
  - Kprobes - Dynamic event tracing in kernel
    - What's dynamic? - trace events in the function body
  - Uprobes - Dynamic event tracing in user space
    - Under development
Kprobe-Tracer

- Dynamic event tracer
  - Based on kprobes (kprobe and kretprobe)
  - Add/delete new events on the fly
  - trace-event/perf-event compatible
    - Enable/disable, filter and record by ftrace and perf tools
- Put a new trace-event with register/memory arguments
  - Function entry (symbol) + offset / function return
  - Fetch various registers/memory/symbols
    - Dereferencing(resolving pointer) is also supported
Kprobe-Tracer: Installation

- Get the latest -tip tree
- Make menuconfig
  - Kernel hacking
    - Tracers (CONFIG_FTRACE = y)
      - Enable kprobes-based dynamic events (CONFIG_KPROBE_EVENT = y)
- Rebuild & install kernel & reboot

- Supported architecture
  - x86/x86-64
  - s390
  - PPC
Kprobe-Tracer: Usage

- See Documentation/trace/kprobetrace.txt

- Interface
  - (debugfs)/tracing/kprobe_events
    - **Write** event definitions
      - echo "command" >> tracing/kprobe_events
        (Note: write without O_APPEND (e.g '>' clears all existing events)
    - **Read** current event definitions
      - cat tracing/kprobe_events
  - (debugfs)/tracing/kprobe_profile
    - Check the profile of each events (nhits/nmissed)

Command)
- `r[:[GRP/]EVENT] SYMBOL[+0] [FETCHARGS]`: Set a return probe
- `-:[GRP/]EVENT`: Clear a probe
Event arguments can access registers/memory/stack

- `%REG` : Fetch register REG
- `@ADDR` : Fetch memory at ADDR (ADDR should be in kernel)
- `@SYM[+|-offs]` : Fetch memory at SYM +|- offs (SYM should be a data symbol)
- `$stackN` : Fetch Nth entry of stack (N >= 0)
- `$stack` : Fetch stack address.
- `$retval` : Fetch return value. (*)
- `+|-offs(FETCHARG)` : Fetch memory at FETCHARG +|- offs address. (**) 
- `NAME=FETCHARG` : Set NAME as the argument name of FETCHARG.
- `FETCHARG:TYPE` : Set TYPE as the type of FETCHARG. Currently, basic types (u8/u16/u32/u64/s8/s16/s32/s64) are supported.

E.g.
- `'foo=+10(%bp):u32'`
  fetch u32 value from the address which bp register value plus 10.

- `'bar=@tick_usec'`
  fetch unsigned long value of tick_usec symbol.
kprobes now checks instruction boundary.

- If a probe puts at the middle or end of a instruction, return -EILSQ
- x86: instruction decoder decodes target function

x86 insn decoder

- Support both of x86/x86-64
  - Support AVX instructions too
- Easy to maintain: generates attribute maps from x86 opcode map
Kprobe-Tracer: Demo

- Probe setting and tracing on vfs_read

```bash
<Analyze Binary>
# grep vfs_read /proc/kallsyms
# objdump -Sd vmlinux --start-address=0x.... | less

<Add Event>
# echo "p vfs_read+.. %di +0x3c(%di):u32" >> kprobe_events

>Show Event>
# cat events/kprobes/p_vfs_read../format
# cat kprobe_events

<Trace Event>
# echo 1 > events/kprobes/p_vfs_read../enable
# cat trace

>Delete Event>
# echo '- p_vfs_read..' >> kprobe_events
```
**Flexible, Dynamic, but Painful**

- **Probepoint**: symbol+offset
  - No source code lines, no inlined functions
  - Objdump helps a bit
- **Argument**: registers/memory
  - No local variables
  - Objdump can't help it
- Users have to disassemble binary and analyze it.

```bash
$ objdump -Sd kernel/sched.o
...
static void update_min_vruntime(struct cfs_rq *cfs_rq)
{
    u64 vruntime = cfs_rq->min_vruntime;
    44b2:       49 8b 45 20             mov    0x20(%r13),%rax
    if (cfs_rq->curr)
    44b6:       48 85 d2                test   %rdx,%rdx
    44b9:       48 89 c1                mov    %rax,%rcx
    ...
```
What Can Help?

- Some tools can support source-code level analysis
  - *Debugger*(gdb)
  - *SystemTap*
- Both use **debuginfo**
  - Debuginfo provides the information of probe points and local variables
    - Source code information
    - Variable/Structure type information
- Analyzing debuginfo requires user space helper
  - Perf-tools
    - A tool in kernel tree
    - Synchronously update with kernel

-> **Perf probe subcommand**
Perf Probe

- Dynamic event control helper
  - Add new trace events on kprobe-tracer from source-code level information
    - Find inline functions / function relative lines
    - Find local variable locations/types
  - Delete those trace events by name
  - List all trace events with source lines
  - Help user to find which lines can be probed

(See tools/perf/Documentation/perf-probe.txt)
Perf Probe: Demo

- Probe setting and tracing on vfs_read

```bash
<Analyze Binary>
# perf probe --line vfs_read

>Add Event>
# perf probe --add 'vfs_read file file->f_mode'

>Show Event>
# perf probe --list
# perf list

Trace Event>
# perf record -e probe:vfs_read -aRf ls -l
# perf trace

>Delete Event>
# perf probe --del '*'
```

We don't see any registers/memory address, or byte-offsets!
• --line shows which source code lines can be probed

Syntax)
perf probe --line FUNCTION[:RelNumber[+NumLINES|-EndNumber]]
perf probe --line SOURCE:AbsNumber[+NumLINES|-EndNumber]

Example)
# perf probe --line vfs_read:0+7
<vfs_read:0>
    ssize_t vfs_read(struct file *file, char __user *buf, size_t
1    {                          
        ssize_t ret;
    
4    if (!(file->f_mode & FMODE_READ))
        return -EBADF;
6    if (!file->f_op || (!file->f_op->read && !file->f_op-

Lines start with number can be probed.
Perf Probe: --add

- --add adds a new event

```
# perf probe --add 'EVENT=PROBE_POINT [ARG1 ARG2 ...]'  
# perf probe 'EVENT=PROBE_POINT [ARG1 ARG2 ...]'  
```

- Event name
  - This will be created from the probed function name
- Probe point
  - Function or File and Line number. Lazy matching is also supported
- Argument
  - Function local variables
  - Kprobe-tracer syntax is also supported
Perf Probe: Probe Points

- Probe point specifies where new event happens

Syntax:

\[
\text{[EVENT=} \text{FUNC[@SRC]}[+\text{Offset}|\%\text{return}|:\text{RelNumber}|;\text{Pattern}]
\]

or

\[
\text{[EVENT=} \text{SOURCE}:\text{AbsNumber}|\text{SOURCE};\text{Pattern}
\]

- Function name base
  - Support inline function
  - Function relative offset / line-number
  - Support function exit (%return)
    - Note that this is only for non-inlined functions
- Source file base
  - Tail matching: “sched.c” matches “…/kernel/sched.c”
- Lazy matching
  - Source line pattern can be specified
Perf Probe: Lazy Matching

- Lazy matching
  - Put events on every line which matches with the pattern
  - Lazy pattern likes a glob('*','?','[]'), but ignores spaces

E.g.
```bash
# perf probe --add 'schedule;cpu=*'
...
# perf probe --list
probe:schedule (on schedule:9@linux-2.6-tip/kernel/sched.c)
probe:schedule_1 (on schedule:55@linux-2.6-tip/kernel/sched.c)
# perf probe --line schedule
[...]
  9     cpu = smp_processor_id();
[...]
  55     cpu = smp_processor_id();
```
• Arguments of events
  • Local variables are translated by using debuginfo
    – Data structure is going to be available
  • Name is set from the variable name
    – Data structure members has another rule – last field name
  • Type casting is going to be supported (u8/16/32/64, s8/16/32/64)

  e.g.
  'count'
  Get a local variable named 'count' (argument name is 'count')

  'file->f_mode'
  Get 'f_mode' member of 'file' local variable as 'f_mode' argument.
Perf Probe: --list

- --list shows current events with source code line numbers
  - Note: arguments are shown by name

```
e.g.
# perf probe --list
  probe:schedule  (on schedule:36@linux-2.6-tip/kernel/sched.c with rq)
  probe:vfs_read  (on vfs_read@linux-2.6-tip/fs/read_write.c with file)
```
Perf Probe: --del

- --del deletes events matching a given glob pattern
- glob expression can be used in other commands (e.g. perf-record)

E.g.

```
# perf probe --del 'schedule*'
```

Remove dynamic events which name start with 'schedule'

```
# perf probe --del '*'
```

Remove ALL dynamic events.
Perf Probe: Options

- **--force**
  - Forcibly add new events on the function in where there are already other events
  - Event name will be “function_N” (N is an index)

- **--dry-run**
  - Don't change kprobe-tracer
  - Only --add/--del are affected

- **--verbose**
  - Show more messages
• **Don't Forget you're on the command-line!**
  
  • Special characters can be translated by shell
    - Kprobe-tracer syntax includes `$`
    - Perf probe syntax includes `';' ']' '*'`
  
  • Using `'` (single-quote) is recommended

• **Test before executing**
  
  • `-fnv` (force, dry-run, verbose) is recommend
Perf Probe: Requirements

- **Kernel built options**
  - Enabling dynamic perf/trace event
    - CONFIG_KPROBE_EVENT
    - CONFIG_PERF_EVENT
  - Building kernel with debuginfo
    - CONFIG_DEBUG_INFO
    - Will get a bigger binary ... don't upset :)

- **Elfutils (Libdw)**
  - Dwarf format (debuginfo) analysis library
    - Developed closely with GCC.
  - Without elfutils, perf probe can't support debuginfo

- **Architecture**
  - x86/x86-64
  - PPC is proposed.
Perf Probe: TODOs

- **Opened TODOs**
  - **String type support**
    - String allows us to trace pathname etc
  - **Module support**
    - Kernel modules are not supported yet
    - Modules can be relocatable
  - **Dynamic indexed array**
    - array[i] is commonly used in loops
  - **%next**
    - Probe the next step line, or just use post_handler
Changelog

- 2.6.33
  - Kprobe-tracer
  - Perf probe: prototype feature
  - Note: Requires libdwarf

- 2.6.34 (expecting)
  - Adding --line/lazy matching support
  - Note: Move onto elfutils (from libdwarf)
    - Elfutils works better with newer gcc
  - Jump optimized kprobes

- -tip (ongoing)
  - Data structure member support
  - Type support
Kprobes Jump Optimization (x86)

- Kprobes enhancement feature by replacing a breakpoint with a jump instruction.

![Diagram showing Kprobes and optimized Kprobes flow]

- Kprobes
  - int3
  - Die notifier
  - Single step
  - Post Kprobes

- Optimized Kprobes
  - Relative jump
  - Jump in
  - Emulate int3/Handler call
  - Direct execution
  - Jump back

Optimize
Performance Improvements

- Overheads/probe (usec) : smaller is better

![Graph showing performance improvements for Kprobes and Kretprobes on Xeon 2.33Ghz](image-url)
Kprobes Jump Optimization: Usage

• Optimization is **transparently** done
  • Don't require any user-side changes
  • No kABI change
    - Just add a flag bit on internal kprobe->flags
  • Not all probes are optimized

• Sysctl interface
  • Disabling/Enabling optimization via sysctl “debug.kprobes-optimization”
    - Enabling (default): debug.kprobes-optimization = 1
    - Disabling: debug.kprobes-optimization = 0
Conclusion

- Dynamic event tracing
  - In-kernel flexible probe framework
  - Events can trace registers/memory
  - Safety checks can check the instruction boundary
- Perf probe
  - Debuginfo analyzer for helping dynamic event setting from source code info
  - User friendly interface for dynamic event tracing
  - In kernel tree tool
- Kprobe jump optimization
  - Reduce kprobe's overhead drastically
  - No user change: Transparently optimized
Related Articles

• LWN.net
  • Dynamic probes with ftrace (kprobe-tracer)
    – http://lwn.net/Articles/343766/
  • Minimizing instrumentation impacts (kprobes jump optimization)
    – http://lwn.net/Articles/365833/
Thank You
Questions/Discussion
• Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

• Other company, product, or service names may be trademarks or service marks of others.