Linux Tracing Tools

Perf and BCC (eBPF)

Ravi Bangoria
ravi.bangoria@linux.ibm.com

Sandipan Das
sandipan@linux.ibm.com

Linux Technology Center
India Systems Development Lab, IBM

Where do you need **Tracing**?

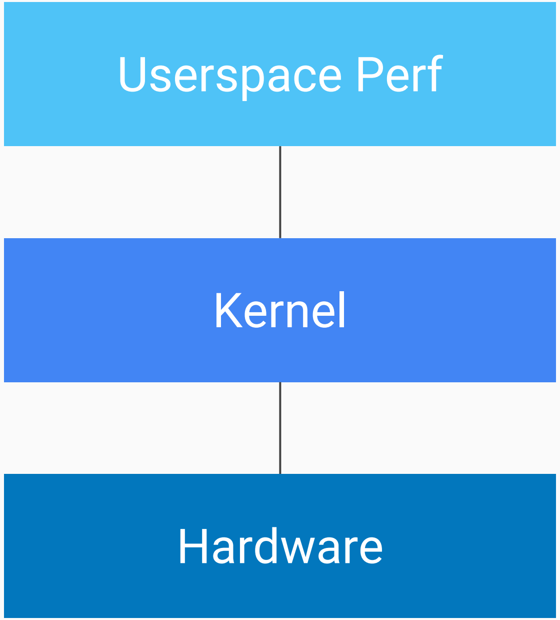
Perf

Why Perf?

- Low overhead
- Easy to use
- Production-safe
- Feature rich
- No service daemons are needed
- Doesn’t stop the workload
- No need to recompile the workload

Agenda

- Performance Monitoring
  - Counting events
  - Profiling (Sampling)
  - Callgraph
  - Drill down to instruction level
  - Perf script
  - Real time profiling
- Static Tracing
- Dynamic Tracing (Kprobe/Uprobe)
- Hardware Breakpoints
- BCC (eBPF)

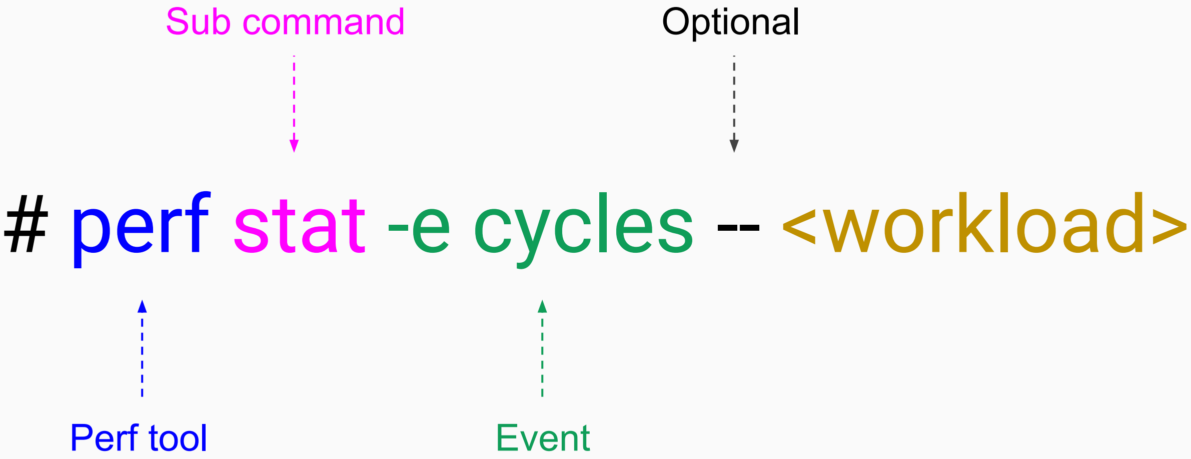


Userspace Perf

Kernel

Hardware

Counting events



Sub command

Optional

```
# perf stat -e cycles -- <workload>
```

Perf tool

Event

Profiling (Sampling)

```
# perf record <workload>
# perf report
```

- Record => perf.data => Report
- Default event is cycles. Use -e for other events
- Default frequency is 4k samples/second
- Record with -a for systemwide, -p for pid specific, -C for CPU specific
- Supports cross arch record / report

Symbol table / Debuginfo

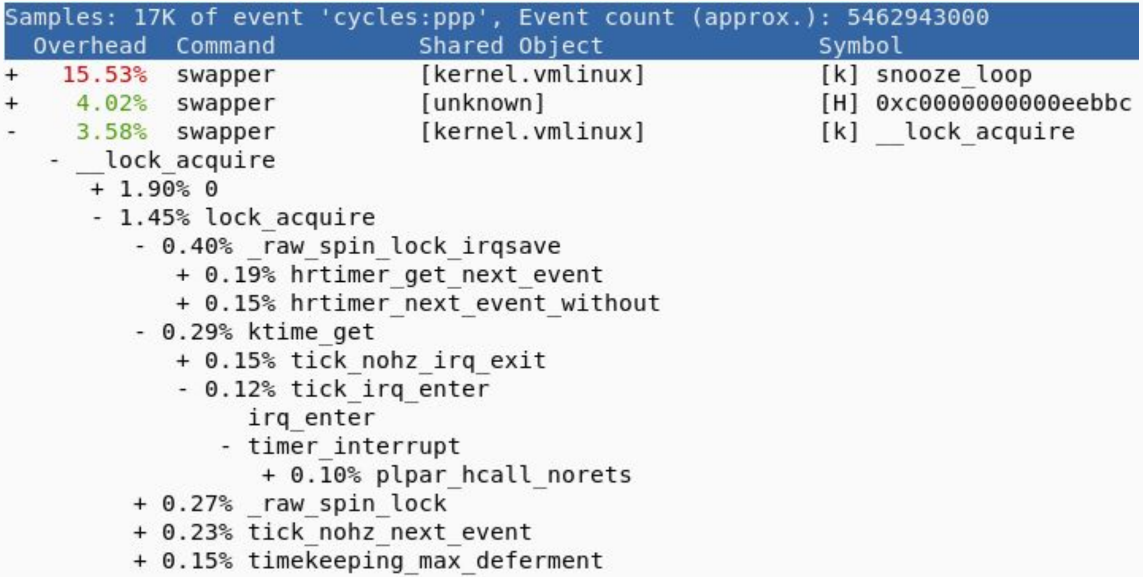
```
# readelf -SW <binary>
```

- Map instruction pointer to symbol name
- Install debuginfo package if distro provided binary is stripped
- Does this mean I’ve to install debuginfo packages on production system? -- NO.

Callgraph

# perf record -g …

# perf report --no-children

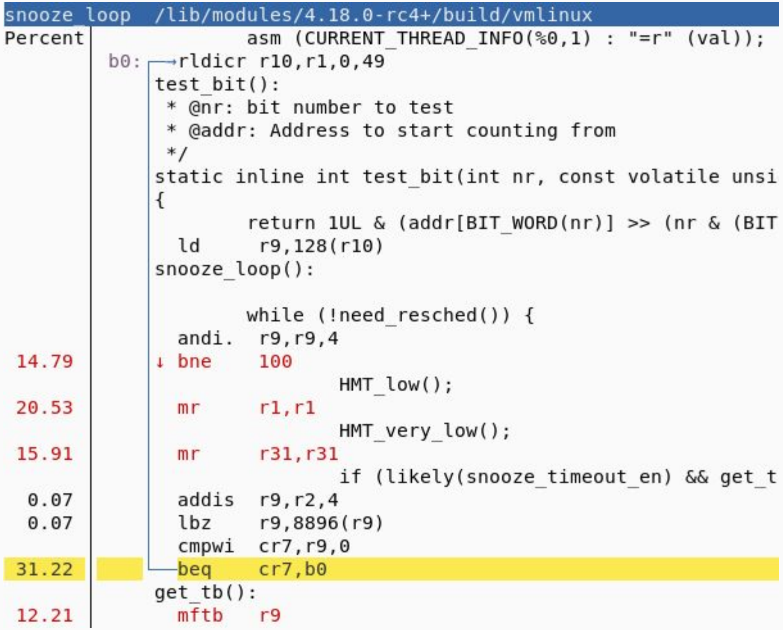


```
Samples: 17K of event 'cycles:ppp', Event count (approx.): 5462943000
  Overhead  Command          Shared Object              Symbol
+   15.53%  swapper          [kernel.vmlinux]           [k] snooze_loop
+    4.02%  swapper          [unknown]                  [H] 0xc0000000000eebbc
-    3.58%  swapper          [kernel.vmlinux]           [k] __lock_acquire
   - __lock_acquire
      + 1.90% 0
      - 1.45% lock_acquire
         - 0.40% _raw_spin_lock_irqsave
            + 0.19% hrtimer_get_next_event
            + 0.15% hrtimer_next_event_without
         - 0.29% ktime_get
            + 0.15% tick_nohz_irq_exit
            - 0.12% tick_irq_enter
                 irq_enter
               - timer_interrupt
                  + 0.10% plpar_hcall_norets
         + 0.27% _raw_spin_lock
         + 0.23% tick_nohz_next_event
         + 0.15% timekeeping_max_deferment
```

Drilldown at instruction level

# perf annotate

- With / Without Source
- Interactive
- Dependencies of instructions
- Read my blog: https://www.ibm.com/developerworks/library/l-analyzing-performance-perf-annotate-trs/index.html



```
snooze_loop  /lib/modules/4.18.0-rc4+/build/vmlinux
Percent│             asm (CURRENT_THREAD_INFO(%0,1) : "=r" (val));
       │ b0: ┌─→rldicr r10,r1,0,49
       │     │test_bit():
       │     │ * @nr: bit number to test
       │     │ * @addr: Address to start counting from
       │     │ */
       │     │static inline int test_bit(int nr, const volatile unsi
       │     │{
       │     │        return 1UL & (addr[BIT_WORD(nr)] >> (nr & (BIT
       │     │  ld     r9,128(r10)
       │     │snooze_loop():
       │     │
       │     │        while (!need_resched()) {
       │     │  andi.  r9,r9,4
  14.79│     │↓ bne    100
       │     │                HMT_low();
  20.53│     │  mr     r1,r1
       │     │                HMT_very_low();
  15.91│     │  mr     r31,r31
       │     │                if (likely(snooze_timeout_en) && get_t
   0.07│     │  addis  r9,r2,4
   0.07│     │  lbz    r9,8896(r9)
       │     │  cmpwi  cr7,r9,0
  31.22│     └──beq    cr7,b0
       │      get_tb():
  12.21│        mftb   r9
```

Perf script

```
# perf script
or
# perf script record/report <script> -- <workload>
```

- Read perf.data and display trace output
- Python and Perl support
- Use --list option for available scripts
- You can write your own script
- Use -g to generate new template from perf.data

Real Time Profiling

```
# perf top
```

- Generate and display profile in real time
- “Top” like but more detailed
- Supports call-graphs, annotate etc. feature in real time
- Interactive

Static Tracing (Tracepoints)

- Profiling takes samples. Tracing records every event.
- Tracepoints are added by developer at important places in the code.
- Kernel tracepoints are already supported.
- Userspace tracepoints(USDT) are partially supported. We are working on providing full support.
- When not being traced they are just “nop”.
- Each tracepoints has their own list of arguments and output format.

Dynamic Tracing (Kprobes)

```
# perf probe <probe_location>
# perf record -e probe:...
```

- Create dynamic tracepoints in kernel
- Probe with arguments, variables
- Return probes
- Probe inside kernel module
- Probe location can be a function, file:lineno, function+offset …
- Need debuginfo for some of the features

Dynamic Tracing (Uprobes)

```
# perf probe -x binary <probe_location>
# perf record -e probe:...
```

- Create dynamic tracepoints in userspace application
- Probe with arguments, variables
- Return probes
- Probe location can be a function, file:lineno, function+offset …
- Need debuginfo for some of the features

Hardware Breakpoints

```
# perf record -e mem:0xc0000000011ea98c …
```

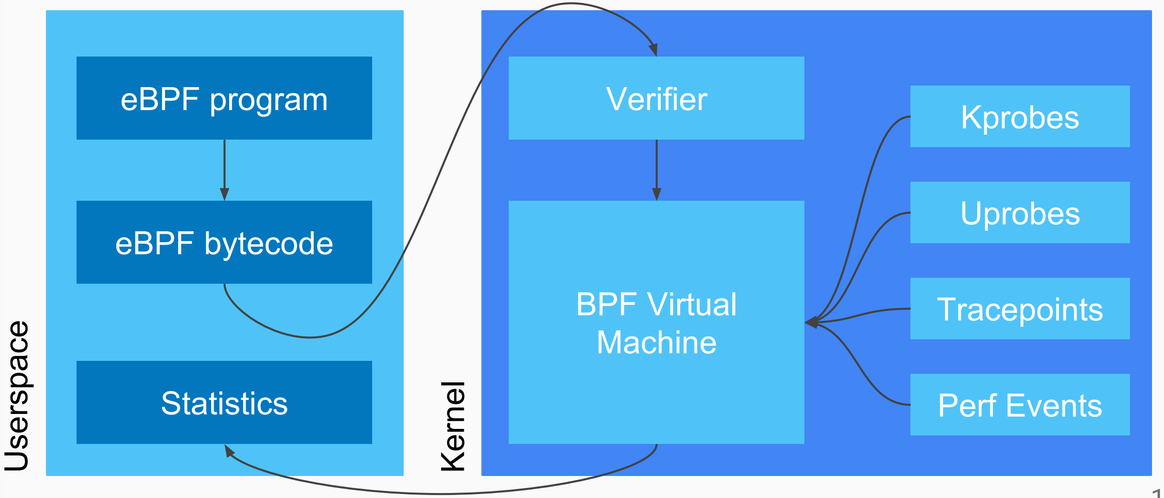
- Types of hw-breakpoints:
    - Data breakpoints (Watchpoints)
    - Instruction breakpoints
- Watchpoint: Who is changing a particular memory location?
- Watchpoints useful for debugging memory corruption problems.
- Instruction breakpoints is same as kprobes but supported by hw.
- Very limited hw registers.

Linux Tracing with eBPF

eBPF

What is eBPF?

- Extended Berkeley Packet Filters
  - But the tracing infrastructure can also exploit it



eBPF

Why eBPF?

- Minimal overhead
- Programmability
  - Compute customized event statistics
  - Perform in-kernel data aggregation
- Production-safe
  - Checks for unsafe code before execution
  - Execution in a secure VM
- Maps
  - Maintain state across events
  - Exchange data with Userspace

BCC

What is BCC?

- eBPF Compiler Collection
  - Toolkit for creating eBPF-based tracing scripts
- Provides a Python API
  - Simplifies tasks
    - Event creation
    - Compilation of eBPF programs
    - Loading and attaching eBPF programs
    - Access to maps
- Set of useful, readily-available tools
  - Trace a variety of kernel subsystems
  - Trace typical enterprise applications

BCC

Demos

- Attaching probes to a function
  - Print arguments
  - Print return value
- Filtering data by process
  - PID filter to capture data for a specific process
- Filtering data by setting thresholds
  - Find approximate execution time of a function
  - Find out when a function runs slower than a given threshold

https://github.com/sandip4n/devconf.in-2018

Contact Us

Ravi Bangoria
ravi.bangoria@linux.ibm.com

Sandipan Das
sandipan@linux.ibm.com

Thank You !

References -- Perf

- Perf tool man pages
- perf_event_open() man page
- *perf Examples* by Brendan Gregg
  http://www.brendangregg.com/perf.html
- Perf wiki
  https://perf.wiki.kernel.org/
- *The Unofficial Linux Perf Events Web-Page* by Vince Weaver
  http://web.eece.maine.edu/~vweaver/projects/perf_events/

References -- BCC and eBPF

- *A thorough introduction to eBPF* by Matt Fleming
  https://lwn.net/Articles/740157/
- *BPF - in-kernel virtual machine* by Alexei Starovoitov
  https://events.linuxfoundation.org/sites/events/files/slides/bpf_collabsummit_2015feb20.pdf
- *Linux BPF Superpowers* by Brendan Gregg
  http://www.slideshare.net/brendangregg/linux-bpf-superpowers
- *Linux Enhanced BPF (eBPF) Tracing Tools* by Brendan Gregg
  http://www.brendangregg.com/ebpf.html
- *BPF: tracing and more* by Brendan Gregg
  http://www.slideshare.net/brendangregg/bpf-tracing-and-more

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of the employers (IBM Corporation).
- IBM and IBM (Logo) are trademarks or registered trademarks of International Business Machines in United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product and service names may be trademarks or service marks of others.

Linux Tracing Tools -- Backup

- Perf
- BCC (eBPF)
- Strace
- Ftrace
- Systemtap
- LTTng
- PCP
- Gprof
- Oprofile