Linux 性能优化工具简介

作者：黄金华

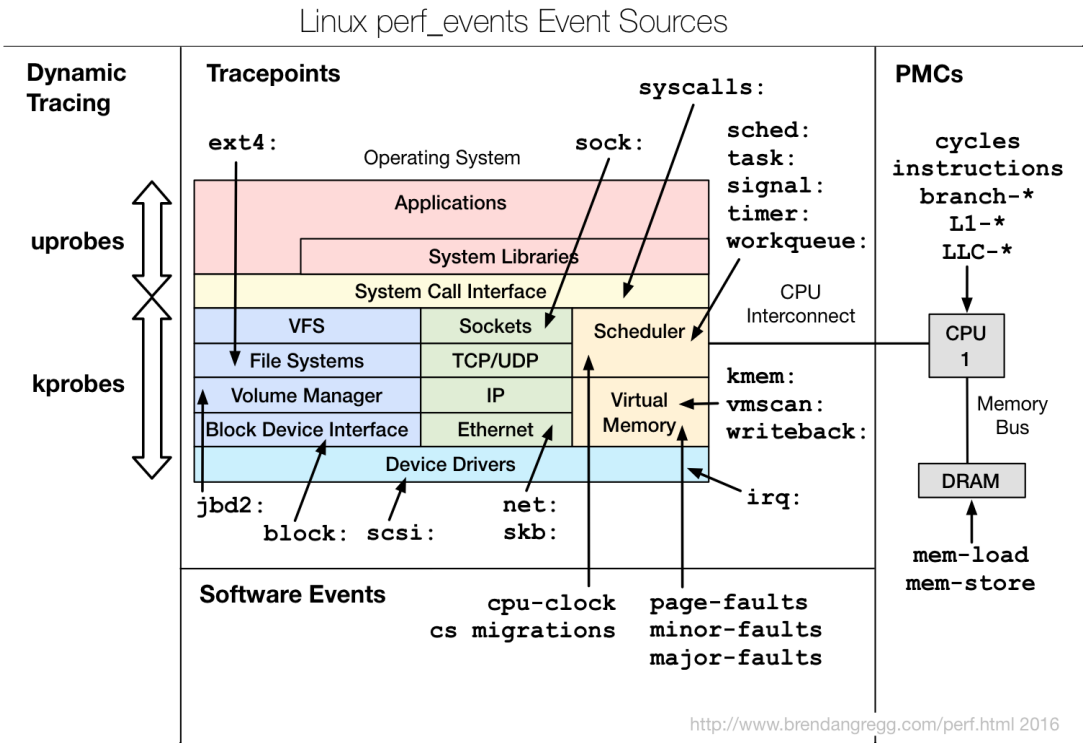
https://github.com/sjtuhjh/perftools

1 Perf(Perf-Event)

1.1 简介

Perf 是 Linux 事件驱动的系统性能分析工具。至于 Linux 事件，目前主要有以下几类：

- Hardware Events (PMC): 硬件寄存器保存的相关性能计数器，例如 cache 命中次数等；
- Software Events: 内核维护的计数器，例如上下文切换次数、软中断次数等；
- Kernel Tracepoint Events: 内核静态跟踪事件（在编译阶段已经插入相关跟踪代码）
- USDT (User Statically-Defined Tracing): 用户程序中静态跟踪事件；
- Dynamic Tracing: 动态跟踪事件（内核态基于 Kprobe，而用户程序基于 Uprobe）。相对于静态跟踪，动态跟踪在编译阶段本身不插入任何代码，只是在运行时根据需要动态插入软中断代码来触发跟踪处理。



具体来说，Perf 通过三种方式对性能事件进行处理：

- 统计：只是对性能事件的统计计数（不会产生 perf.data）
- 采样：以一定频率（可以使用-F 指定频率）对系统进行采样，并将相关数据写入 perf.data，最后通过 perf report 或 perf script 进行读取
- bpf 程序（关于 bpf 本身，请参考第三节）：支持向内核传递用户自定义的 bpf 代码处理相关性能事件； 另外，目前最新的 perf 工具本身（kernel 4.9+）部分功能就已经依赖 bpf。

1.2 安装

- 编译/安装：
  在内核代码的 tools/perf 下编译安装即可（或者通过安装 linux-tools-*包）

- 内核编译选项：

```
# for perf_events:
CONFIG_PERF_EVENTS=y
# for stack traces:
CONFIG_FRAME_POINTER=y
# kernel symbols:
CONFIG_KALLSYMS=y
# tracepoints:
CONFIG_TRACEPOINTS=y
# kernel function trace:
CONFIG_FTRACE=y
# kernel-level dynamic tracing:
CONFIG_KPROBES=y
CONFIG_KPROBE_EVENTS=y
# user-level dynamic tracing:
CONFIG_UPROBES=y
CONFIG_UPROBE_EVENTS=y
# full kernel debug info:
CONFIG_DEBUG_INFO=y
# kernel lock tracing:
CONFIG_LOCKDEP=y
# kernel lock tracing:
CONFIG_LOCK_STAT=y
# kernel dynamic tracepoint variables:
CONFIG_DEBUG_INFO=y
```

- 支持内核符号解析：
  echo 0 > /proc/sys/kernel/kptr_restrict

1.3 使用案例

1.3.1 列出事件

1.3.2 统计计数

```
# perf stat -d gzip file1

 Performance counter stats for 'gzip file1':

      1610.719530 task-clock                #    0.998 CPUs utilized
               20 context-switches          #    0.012 K/sec
                0 CPU-migrations            #    0.000 K/sec
              258 page-faults               #    0.160 K/sec
    5,491,605,997 cycles                    #    3.409 GHz                      [40.18%]
    1,654,551,151 stalled-cycles-frontend   #   30.13% frontend cycles idle     [40.80%]
    1,025,280,350 stalled-cycles-backend    #   18.67% backend  cycles idle     [40.34%]
    8,644,643,951 instructions              #    1.57  insns per cycle
                                            #    0.19  stalled cycles per insn [50.89%]
    1,492,911,665 branches                  #  926.860 M/sec                    [50.69%]
       53,471,580 branch-misses             #    3.58% of all branches          [51.21%]
    1,938,889,736 L1-dcache-loads           # 1203.741 M/sec                    [49.68%]
      154,380,395 L1-dcache-load-misses     #    7.96% of all L1-dcache hits    [49.66%]
                0 LLC-loads                 #    0.000 K/sec                    [39.27%]
                0 LLC-load-misses           #    0.00% of all LL-cache hits     [39.61%]

      1.614165346 seconds time elapsed
```

```
# perf list | grep L1-dcache
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetches                               [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
# perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores gzip file1

 Performance counter stats for 'gzip file1':

     1,947,551,657 L1-dcache-loads

       153,829,652 L1-dcache-misses
         #    7.90% of all L1-dcache hits
     1,171,475,286 L1-dcache-stores


       1.538038091 seconds time elapsed
```

```
# perf stat -e cycles,instructions,r80a2,r2b1 gzip file1

 Performance counter stats for 'gzip file1':

     5,586,963,328 cycles                    #    0.000 GHz
     8,608,237,932 instructions              #    1.54  insns per cycle
         9,448,159 raw 0x80a2
    11,855,777,803 raw 0x2b1

       1.588618969 seconds time elapsed
```

（注释：r80a2 等是寄存器名，与具体 CPU 型号相关）

```
# CPU counter statistics for the specified command:
perf stat command

# Detailed CPU counter statistics (includes extras) for the specified command:
perf stat -d command

# CPU counter statistics for the specified PID, until Ctrl-C:
perf stat -p PID

# CPU counter statistics for the entire system, for 5 seconds:
perf stat -a sleep 5

# Various basic CPU statistics, system wide, for 10 seconds:
perf stat -e cycles,instructions,cache-references,cache-misses,bus-cycles -a sleep 10

# Various CPU level 1 data cache statistics for the specified command:
perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores command

# Various CPU data TLB statistics for the specified command:
perf stat -e dTLB-loads,dTLB-load-misses,dTLB-prefetch-misses command

# Various CPU last level cache statistics for the specified command:
perf stat -e LLC-loads,LLC-load-misses,LLC-stores,LLC-prefetches command

# Using raw PMC counters, eg, unhalted core cycles:
perf stat -e r003c -a sleep 5

# PMCs: cycles and frontend stalls via raw specification:
perf stat -e cycles -e cpu/event=0x0e,umask=0x01,inv,cmask=0x01/ -a sleep 5
```

```
# Count system calls for the specified PID, until Ctrl-C:
perf stat -e 'syscalls:sys_enter_*' -p PID

# Count system calls for the entire system, for 5 seconds:
perf stat -e 'syscalls:sys_enter_*' -a sleep 5

# Count scheduler events for the specified PID, until Ctrl-C:
perf stat -e 'sched:*' -p PID

# Count scheduler events for the specified PID, for 10 seconds:
perf stat -e 'sched:*' -p PID sleep 10

# Count ext4 events for the entire system, for 10 seconds:
perf stat -e 'ext4:*' -a sleep 10

# Count block device I/O events for the entire system, for 10 seconds:
perf stat -e 'block:*' -a sleep 10

# Count all vmscan events, printing a report every second:
perf stat -e 'vmscan:*' -a -I 1000

# Show system calls by process, refreshing every 2 seconds:
perf top -e raw_syscalls:sys_enter -ns comm

# Show sent network packets by on-CPU process, rolling output (no clear):
stdbuf -oL perf top -e net:net_dev_xmit -ns comm | strings
```

Special

```
# Record cacheline events (Linux 4.10+):
perf c2c record -a -- sleep 10

# Report cacheline events from previous recording (Linux 4.10+):
perf c2c report
```

Reporting

```
# Show perf.data in an ncurses browser (TUI) if possible:
perf report

# Show perf.data with a column for sample count:
perf report -n

# Show perf.data as a text report, with data coalesced and percentages:
perf report --stdio

# Report, with stacks in folded format: one line per stack (needs 4.4):
perf report --stdio -n -g folded

# List all events from perf.data:
perf script

# List all perf.data events, with data header (newer kernels; was previously default):
perf script --header

# List all perf.data events, with customized fields (< Linux 4.1):
perf script -f time,event,trace

# List all perf.data events, with customized fields (>= Linux 4.1):
perf script -F time,event,trace

# List all perf.data events, with my recommended fields (needs record -a; newer kernels):
perf script --header -F comm,pid,tid,cpu,time,event,ip,sym,dso

# List all perf.data events, with my recommended fields (needs record -a; older kernels):
perf script -f comm,pid,tid,cpu,time,event,ip,sym,dso

# Dump raw contents from perf.data as hex (for debugging):
perf script -D

# Disassemble and annotate instructions with percentages (needs some debuginfo):
perf annotate --stdio
```

1.3.3 采样

```
# perf record -F 99 -a -g -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 3.135 MB perf.data (~136971 samples) ]
# ls -lh perf.data
-rw------- 1 root root 3.2M Jan 26 07:26 perf.data
```

```
# perf record -e L1-dcache-load-misses -c 10000 -ag -- sleep 5
```

```
# Sample on-CPU functions for the specified command, at 99 Hertz:
perf record -F 99 command

# Sample on-CPU functions for the specified PID, at 99 Hertz, until Ctrl-C:
perf record -F 99 -p PID

# Sample on-CPU functions for the specified PID, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID sleep 10

# Sample CPU stack traces for the specified PID, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID -g -- sleep 10

# Sample CPU stack traces for the PID, using dwarf to unwind stacks, at 99 Hertz, for 10 seconds:
perf record -F 99 -p PID -g dwarf sleep 10

# Sample CPU stack traces for the entire system, at 99 Hertz, for 10 seconds (< Linux 4.11):
perf record -F 99 -ag -- sleep 10

# Sample CPU stack traces for the entire system, at 99 Hertz, for 10 seconds (>= Linux 4.11):
perf record -F 99 -g -- sleep 10

# If the previous command didn't work, try forcing perf to use the cpu-clock event:
perf record -F 99 -e cpu-clock -ag -- sleep 10
```

```
# Sample CPU stack traces for the entire system, with dwarf stacks, at 99 Hertz, for 10 seconds:
perf record -F 99 -ag dwarf sleep 10

# Sample CPU stack traces, once every 10,000 Level 1 data cache misses, for 5 seconds:
perf record -e L1-dcache-load-misses -c 10000 -ag -- sleep 5

# Sample CPU stack traces, once every 100 last level cache misses, for 5 seconds:
perf record -e LLC-load-misses -c 100 -ag -- sleep 5

# Sample on-CPU kernel instructions, for 5 seconds:
perf record -e cycles:k -a -- sleep 5

# Sample on-CPU user instructions, for 5 seconds:
perf record -e cycles:u -a -- sleep 5

# Sample on-CPU user instructions precisely (using PEBS), for 5 seconds:
perf record -e cycles:up -a -- sleep 5

# Perform branch tracing (needs HW support), for 1 second:
perf record -b -a sleep 1

# Sample CPUs at 49 Hertz, and show top addresses and symbols, live (no perf.data file):
perf top -F 49

# Sample CPUs at 49 Hertz, and show top process names and segments, live:
perf top -F 49 -ns comm,dso
```

1.3.4 内核静态跟踪

```
# perf stat -e 'syscalls:sys_enter_*' gzip file1 2>&1 | awk '$1 != 0'

 Performance counter stats for 'gzip file1':

                 1 syscalls:sys_enter_utimensat
                 1 syscalls:sys_enter_unlink
                 5 syscalls:sys_enter_newfstat
             1,603 syscalls:sys_enter_read
             3,201 syscalls:sys_enter_write
                 5 syscalls:sys_enter_access
                 1 syscalls:sys_enter_fchmod
                 1 syscalls:sys_enter_fchown
                 6 syscalls:sys_enter_open
                 9 syscalls:sys_enter_close
                 8 syscalls:sys_enter_mprotect
                 1 syscalls:sys_enter_brk
                 1 syscalls:sys_enter_munmap
                 1 syscalls:sys_enter_set_robust_list
                 1 syscalls:sys_enter_futex
                 1 syscalls:sys_enter_getrlimit
                 5 syscalls:sys_enter_rt_sigprocmask
                14 syscalls:sys_enter_rt_sigaction
                 1 syscalls:sys_enter_exit_group
                 1 syscalls:sys_enter_set_tid_address
                14 syscalls:sys_enter_mmap

       1.543990940 seconds time elapsed
```

```
# Trace new processes, until Ctrl-C:
perf record -e sched:sched_process_exec -a

# Trace all context-switches, until Ctrl-C:
perf record -e context-switches -a

# Trace context-switches via sched tracepoint, until Ctrl-C:
perf record -e sched:sched_switch -a

# Trace all context-switches with stack traces, until Ctrl-C:
perf record -e context-switches -ag

# Trace all context-switches with stack traces, for 10 seconds:
perf record -e context-switches -ag -- sleep 10

# Trace all CS, stack traces, and with timestamps (< Linux 3.17, -T now default):
perf record -e context-switches -ag -T

# Trace CPU migrations, for 10 seconds:
perf record -e migrations -a -- sleep 10

# Trace all connect()s with stack traces (outbound connections), until Ctrl-C:
perf record -e syscalls:sys_enter_connect -ag

# Trace all accepts()s with stack traces (inbound connections), until Ctrl-C:
perf record -e syscalls:sys_enter_accept* -ag

# Trace all block device (disk I/O) requests with stack traces, until Ctrl-C:
perf record -e block:block_rq_insert -ag

# Trace all block device issues and completions (has timestamps), until Ctrl-C:
perf record -e block:block_rq_issue -e block:block_rq_complete -a

# Trace all block completions, of size at least 100 Kbytes, until Ctrl-C:
perf record -e block:block_rq_complete --filter 'nr_sector > 200'
```

```
# Trace all block completions, synchronous writes only, until Ctrl-C:
perf record -e block:block_rq_complete --filter 'rwbs == "WS"'

# Trace all block completions, all types of writes, until Ctrl-C:
perf record -e block:block_rq_complete --filter 'rwbs ~ "*W*"'

# Trace all minor faults (RSS growth) with stack traces, until Ctrl-C:
perf record -e minor-faults -ag

# Trace all page faults with stack traces, until Ctrl-C:
perf record -e page-faults -ag

# Trace all ext4 calls, and write to a non-ext4 location, until Ctrl-C:
perf record -e 'ext4:*' -o /tmp/perf.data -a

# Trace kswapd wakeup events, until Ctrl-C:
perf record -e vmscan:mm_vmscan_wakeup_kswapd -ag

# Add Node.js USDT probes (Linux 4.10+):
perf buildid-cache --add `which node`

# Trace the node http__server__request USDT event (Linux 4.10+):
perf record -e sdt_node:http__server__request -a
```

1.3.5 用户静态跟踪

```
# perf buildid-cache --add `which node`
# perf list | grep sdt_node
  sdt_node:gc__done                                  [SDT event]
  sdt_node:gc__start                                 [SDT event]
  sdt_node:http__client__request                     [SDT event]
  sdt_node:http__client__response                    [SDT event]
  sdt_node:http__server__request                     [SDT event]
  sdt_node:http__server__response                    [SDT event]
  sdt_node:net__server__connection                   [SDT event]
  sdt_node:net__stream__end                          [SDT event]
# perf record -e sdt_node:http__server__request -a
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.446 MB perf.data (3 samples) ]
# perf script
            node  7646 [002]   361.012364: sdt_node:http__server__request: (dc2e69)
            node  7646 [002]   361.204718: sdt_node:http__server__request: (dc2e69)
            node  7646 [002]   361.363043: sdt_node:http__server__request: (dc2e69)
```

1.3.6 动态跟踪

```
# perf probe --add tcp_sendmsg
Failed to find path of kernel module.
Added new event:
  probe:tcp_sendmsg     (on tcp_sendmsg)

You can now use it in all perf tools, such as:

         perf record -e probe:tcp_sendmsg -aR sleep 1
```

```
# perf record -e probe:tcp_sendmsg -a -g -- sleep 5
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.228 MB perf.data (~9974 samples) ]
```

Creating a probe for tcp_sendmsg() with the "size" variable:

```
# perf probe --add 'tcp_sendmsg size'
Added new event:
  probe:tcp_sendmsg     (on tcp_sendmsg with size)

You can now use it in all perf tools, such as:

        perf record -e probe:tcp_sendmsg -aR sleep 1
```

Tracing this probe:

```
# perf record -e probe:tcp_sendmsg -a
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.052 MB perf.data (~2252 samples) ]
# perf script
```

Kernel: tcp_sendmsg() line number and local variable

With debuginfo, perf_events can create tracepoints for lines within kernel functions. Listing available line probes for tcp_sendmsg():

```
# perf probe -L tcp_sendmsg
<tcp_sendmsg@/mnt/src/linux-3.14.5/net/ipv4/tcp.c:0>
      0  int tcp_sendmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
                         size_t size)
      2  {
                 struct iovec *iov;
                 struct tcp_sock *tp = tcp_sk(sk);
                 struct sk_buff *skb;
      6          int iovlen, flags, err, copied = 0;
      7          int mss_now = 0, size_goal, copied_syn = 0, offset = 0;
                 bool sg;
                 long timeo;
[...]
     79                  while (seglen > 0) {
                                int copy = 0;
```

```
# perf probe -V tcp_sendmsg:81
Available variables at tcp_sendmsg:81
        @<tcp_sendmsg+537>
                bool    sg
                int     copied
                int     copied_syn
                int     flags
                int     mss_now
                int     offset
                int     size_goal
                long int        timeo
                size_t  seglen
                struct iovec*   iov
                struct sock*    sk
                unsigned char*  from
```

Now lets trace line 81, with the seglen variable that is checked in the loop:

```
# perf probe --add 'tcp_sendmsg:81 seglen'
Added new event:
  probe:tcp_sendmsg    (on tcp_sendmsg:81 with seglen)

You can now use it in all perf tools, such as:

        perf record -e probe:tcp_sendmsg -aR sleep 1

# perf record -e probe:tcp_sendmsg -a
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.188 MB perf.data (~8200 samples) ]
# perf script
            sshd  4652 [001] 2082360.931086: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0x80
   app_plugin.pl  2400 [001] 2082360.970489: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0x20
        postgres  2422 [000] 2082360.970703: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0x52
   app_plugin.pl  2400 [000] 2082360.970890: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0x7b
        postgres  2422 [001] 2082360.971099: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0xb
   app_plugin.pl  2400 [000] 2082360.971140: probe:tcp_sendmsg: (ffffffff81642ca9) seglen=0x55
[...]
```

Adding a libc malloc() probe:

```
# perf probe -x /lib/x86_64-linux-gnu/libc-2.15.so --add malloc
Added new event:
  probe_libc:malloc    (on 0x82f20)

You can now use it in all perf tools, such as:

        perf record -e probe_libc:malloc -aR sleep 1
```

```
# Add a tracepoint for the kernel tcp_sendmsg() function entry ("--add" is optional):
perf probe --add tcp_sendmsg

# Remove the tcp_sendmsg() tracepoint (or use "--del"):
perf probe -d tcp_sendmsg

# Add a tracepoint for the kernel tcp_sendmsg() function return:
perf probe 'tcp_sendmsg%return'

# Show available variables for the kernel tcp_sendmsg() function (needs debuginfo):
perf probe -V tcp_sendmsg

# Show available variables for the kernel tcp_sendmsg() function, plus external vars (needs debuginfo):
perf probe -V tcp_sendmsg --externs

# Show available line probes for tcp_sendmsg() (needs debuginfo):
perf probe -L tcp_sendmsg

# Show available variables for tcp_sendmsg() at line number 81 (needs debuginfo):
perf probe -V tcp_sendmsg:81

# Add a tracepoint for tcp_sendmsg(), with three entry argument registers (platform specific):
perf probe 'tcp_sendmsg %ax %dx %cx'

# Add a tracepoint for tcp_sendmsg(), with an alias ("bytes") for the %cx register (platform specific):
perf probe 'tcp_sendmsg bytes=%cx'

# Trace previously created probe when the bytes (alias) variable is greater than 100:
perf record -e probe:tcp_sendmsg --filter 'bytes > 100'

# Add a tracepoint for tcp_sendmsg() return, and capture the return value:
perf probe 'tcp_sendmsg%return $retval'

# Add a tracepoint for tcp_sendmsg(), and "size" entry argument (reliable, but needs debuginfo):
perf probe 'tcp_sendmsg size'

# Add a tracepoint for tcp_sendmsg(), with size and socket state (needs debuginfo):
perf probe 'tcp_sendmsg size sk->__sk_common.skc_state'

# Tell me how on Earth you would do this, but don't actually do it (needs debuginfo):
perf probe -nv 'tcp_sendmsg size sk->__sk_common.skc_state'
```

```
# Trace previous probe when size is non-zero, and state is not TCP_ESTABLISHED(1) (needs debuginfo):
perf record -e probe:tcp_sendmsg --filter 'size > 0 && skc_state != 1' -a

# Add a tracepoint for tcp_sendmsg() line 81 with local variable seglen (needs debuginfo):
perf probe 'tcp_sendmsg:81 seglen'

# Add a tracepoint for do_sys_open() with the filename as a string (needs debuginfo):
perf probe 'do_sys_open filename:string'

# Add a tracepoint for myfunc() return, and include the retval as a string:
perf probe 'myfunc%return +0($retval):string'

# Add a tracepoint for the user-level malloc() function from libc:
perf probe -x /lib64/libc.so.6 malloc

# Add a tracepoint for this user-level static probe (USDT, aka SDT event):
perf probe -x /usr/lib64/libpthread-2.24.so %sdt_libpthread:mutex_entry

# List currently available dynamic probes:
perf probe -l
```

1.3.7 调度性能分析

```
# perf sched record -- sleep 1
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.886 MB perf.data (13502 samples) ]
```

```
# perf sched latency

 -----------------------------------------------------------------------------------------------------------------
  Task                  |   Runtime ms  | Switches | Average delay ms | Maximum delay ms | Maximum delay at       |
 -----------------------------------------------------------------------------------------------------------------
  cat:(6)               |     12.002 ms |        6 | avg:   17.541 ms | max:   29.702 ms | max at: 991962.948070 s
  ar:17043              |      3.191 ms |        1 | avg:   13.638 ms | max:   13.638 ms | max at: 991963.048070 s
  rm:(10)               |     20.955 ms |       10 | avg:   11.212 ms | max:   19.598 ms | max at: 991963.404069 s
  objdump:(6)           |     35.870 ms |        8 | avg:   10.969 ms | max:   16.509 ms | max at: 991963.424443 s
  :17008:17008          |    462.213 ms |       50 | avg:   10.464 ms | max:   35.999 ms | max at: 991963.120069 s
  grep:(7)              |     21.655 ms |       11 | avg:    9.465 ms | max:   24.502 ms | max at: 991963.464082 s
  fixdep:(6)            |     81.066 ms |        8 | avg:    9.023 ms | max:   19.521 ms | max at: 991963.120068 s
  mv:(10)               |     30.249 ms |       14 | avg:    8.380 ms | max:   21.688 ms | max at: 991963.200073 s
  ld:(3)                |     14.353 ms |        6 | avg:    7.376 ms | max:   15.498 ms | max at: 991963.452070 s
  recordmcount:(7)      |     14.629 ms |        9 | avg:    7.155 ms | max:   18.964 ms | max at: 991963.292100 s
  svstat:17067          |      1.862 ms |        1 | avg:    6.142 ms | max:    6.142 ms | max at: 991963.280069 s
  cc1:(21)              |   6013.457 ms |     1138 | avg:    5.305 ms | max:   44.001 ms | max at: 991963.436070 s
  gcc:(18)              |     43.596 ms |       40 | avg:    3.905 ms | max:   26.994 ms | max at: 991963.380069 s
  ps:17073              |     27.158 ms |        4 | avg:    3.751 ms | max:    8.000 ms | max at: 991963.332070 s
[...]
```

```
# perf sched map
                      *A0           991962.879971 secs A0 => perf:16999
                       A0     *B0    991962.880070 secs B0 => cc1:16863
          *C0          A0      B0    991962.880070 secs C0 => :17023:17023
  *D0      C0          A0      B0    991962.880078 secs D0 => ksoftirqd/0:6
   D0      C0 *E0      A0      B0    991962.880081 secs E0 => ksoftirqd/3:28
   D0      C0 *F0      A0      B0    991962.880093 secs F0 => :17022:17022
  *G0      C0  F0      A0      B0    991962.880108 secs G0 => :17016:17016
   G0      C0  F0     *H0      B0    991962.880256 secs H0 => migration/5:39
   G0      C0  F0     *I0      B0    991962.880276 secs I0 => perf:16984
   G0      C0  F0     *J0      B0    991962.880687 secs J0 => cc1:16996
   G0      C0 *K0      J0      B0    991962.881839 secs K0 => cc1:16945
   G0      C0  K0      J0 *L0  B0    991962.881841 secs L0 => :17020:17020
   G0      C0  K0      J0 *M0  B0    991962.882289 secs M0 => make:16637
   G0      C0  K0      J0 *N0  B0    991962.883102 secs N0 => make:16545
   G0     *O0  K0      J0  N0  B0    991962.883880 secs O0 => cc1:16819
   G0 *A0  O0  K0      J0  N0  B0    991962.884069 secs
   G0  A0  O0  K0 *P0  J0  N0  B0    991962.884076 secs P0 => rcu_sched:7
```

```
# perf sched timehist
Samples do not have callchains.
           time     cpu  task name                       wait time  sch delay   run time
                        [tid/pid]                          (msec)     (msec)     (msec)
--------------- ------  ------------------------------  ---------  ---------  ---------
  991962.879971 [0005]  perf[16984]                         0.000      0.000      0.000
  991962.880070 [0007]  :17008[17008]                       0.000      0.000      0.000
  991962.880070 [0002]  cc1[16880]                          0.000      0.000      0.000
  991962.880078 [0000]  cc1[16881]                          0.000      0.000      0.000
  991962.880081 [0003]  cc1[16945]                          0.000      0.000      0.000
  991962.880093 [0003]  ksoftirqd/3[28]                     0.000      0.007      0.012
  991962.880108 [0000]  ksoftirqd/0[6]                      0.000      0.007      0.030
  991962.880256 [0005]  perf[16999]                         0.000      0.005      0.285
  991962.880276 [0005]  migration/5[39]                     0.000      0.007      0.019
  991962.880687 [0005]  perf[16984]                         0.304      0.000      0.411
  991962.881839 [0003]  cat[17022]                          0.000      0.000      1.746
  991962.881841 [0006]  cc1[16825]                          0.000      0.000      0.000
```

【注释】：perf sched timehist 从 4.10 开始才支持

```
# perf sched timehist -MVw
Samples do not have callchains.
          time     cpu  012345678  task name          wait time  sch delay   run time
                                   [tid/pid]            (msec)     (msec)     (msec)
--------------- ------  ---------  ----------------  ---------  ---------  ---------
  991962.879966 [0005]             perf[16984]                                        awakened: perf[16999]
  991962.879971 [0005]       s     perf[16984]          0.000      0.000      0.000
  991962.880070 [0007]        s    :17008[17008]        0.000      0.000      0.000
  991962.880070 [0002]    s        cc1[16880]           0.000      0.000      0.000
  991962.880071 [0000]             cc1[16881]                                         awakened: ksoftirqd/0[6]
  991962.880073 [0003]             cc1[16945]                                         awakened: ksoftirqd/3[28]
  991962.880078 [0000]  s          cc1[16881]           0.000      0.000      0.000
  991962.880081 [0003]     s       cc1[16945]           0.000      0.000      0.000
  991962.880093 [0003]     s       ksoftirqd/3[28]      0.000      0.007      0.012
  991962.880108 [0000]  s          ksoftirqd/0[6]       0.000      0.007      0.030
  991962.880249 [0005]             perf[16999]                                        awakened: migration/5[39]
  991962.880256 [0005]       s     perf[16999]          0.000      0.005      0.285
  991962.880264 [0005]        m      migration/5[39]                                    migrated: perf[16999] cpu 5 => 1
```

perf sched script dumps all events (similar to **perf script**):

```
# perf sched script

    perf 16984 [005] 991962.879960: sched:sched_stat_runtime: comm=perf pid=16984 runtime=3901506 [ns] vruntime=165...
    perf 16984 [005] 991962.879966:       sched:sched_wakeup: comm=perf pid=16999 prio=120 target_cpu=005
    perf 16984 [005] 991962.879971:       sched:sched_switch: prev_comm=perf prev_pid=16984 prev_prio=120 prev_stat...
    perf 16999 [005] 991962.880058: sched:sched_stat_runtime: comm=perf pid=16999 runtime=98309 [ns] vruntime=16405...
     cc1 16881 [000] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16881 runtime=3999231 [ns] vruntime=7897...
  :17024 17024 [004] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=17024 runtime=3866637 [ns] vruntime=7810...
     cc1 16900 [001] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16900 runtime=3006028 [ns] vruntime=7772...
     cc1 16825 [006] 991962.880058: sched:sched_stat_runtime: comm=cc1 pid=16825 runtime=3999423 [ns] vruntime=7876...
```

1.3.8 BPF

Here is my BPF program, kca_from.c:

```
#include <uapi/linux/bpf.h>
#include <uapi/linux/ptrace.h>

#define SEC(NAME) __attribute__((section(NAME), used))

/*
 * Edit the following to match the instruction address range you want to
 * sample. Eg, look in /proc/kallsyms. The addresses will change for each
 * kernel version and build.
 */
#define RANGE_START  0xffffffff817c1bb0
#define RANGE_END    0xffffffff8187bd89

struct bpf_map_def {
        unsigned int type;
        unsigned int key_size;
        unsigned int value_size;
        unsigned int max_entries;
};

static int (*probe_read)(void *dst, int size, void *src) =
    (void *)BPF_FUNC_probe_read;
static int (*get_smp_processor_id)(void) =
    (void *)BPF_FUNC_get_smp_processor_id;
static int (*perf_event_output)(void *, struct bpf_map_def *, int, void *,
    unsigned long) = (void *)BPF_FUNC_perf_event_output;

struct bpf_map_def SEC("maps") channel = {
        .type = BPF_MAP_TYPE_PERF_EVENT_ARRAY,
        .key_size = sizeof(int),
        .value_size = sizeof(u32),
        .max_entries = __NR_CPUS__,
};

SEC("func=kmem_cache_alloc")
int func(struct pt_regs *ctx)
{
        u64 ret = 0;
        // x86_64 specific:
        probe_read(&ret, sizeof(ret), (void *)(ctx->bp+8));
        if (ret >= RANGE_START && ret < RANGE_END) {
                perf_event_output(ctx, &channel, get_smp_processor_id(),
                    &ret, sizeof(ret));
        }
        return 0;
}

char _license[] SEC("license") = "GPL";
int _version SEC("version") = LINUX_VERSION_CODE;
```

```
# perf record -e bpf-output/no-inherit,name=evt/ -e ./kca_from.c/map:channel.event=evt/ -a -- sleep 1
bpf: builtin compilation failed: -95, try external compiler
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.214 MB perf.data (3 samples) ]

# perf script
 testserver00001 14337 [003] 481432.395181:          0     evt:  ffffffff81210f51 kmem_cache_alloc (/lib/modules/...)
      BPF output: 0000: 0f b4 7c 81 ff ff ff ff  ..|.....
                  0008: 00 00 00 00              ....

    redis-server  1871 [005] 481432.395258:          0     evt:  ffffffff81210f51 kmem_cache_alloc (/lib/modules/...)
      BPF output: 0000: 14 55 7c 81 ff ff ff ff  .U|.....
                  0008: 00 00 00 00              ....

    redis-server  1871 [005] 481432.395456:          0     evt:  ffffffff81210f51 kmem_cache_alloc (/lib/modules/...)
      BPF output: 0000: fe dc 7d 81 ff ff ff ff  ..}.....
                  0008: 00 00 00 00              ....
```

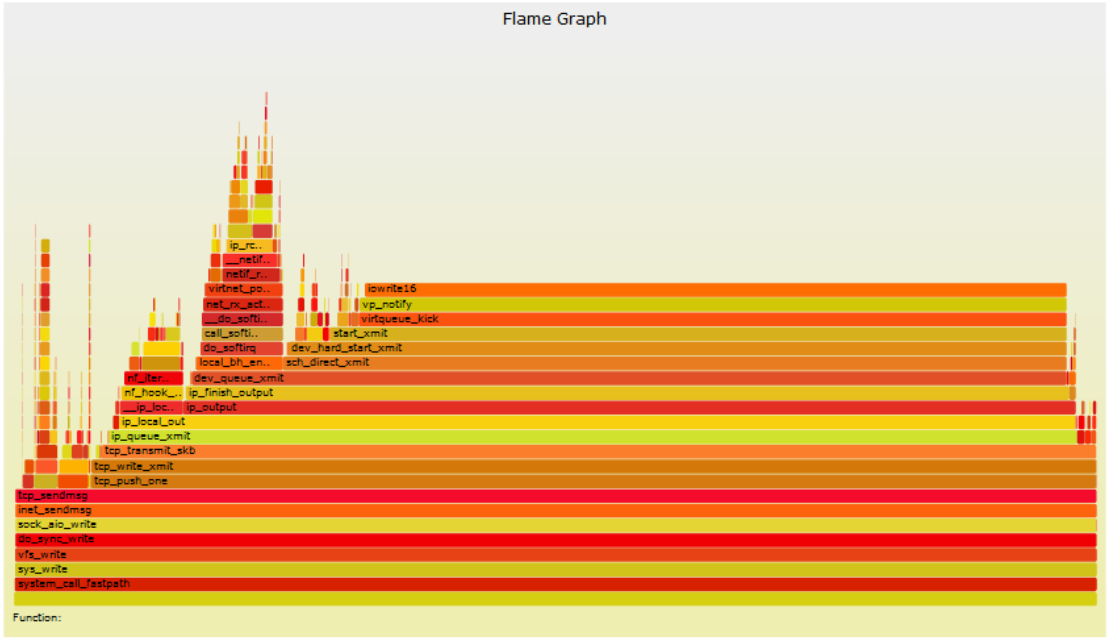
Adding stack traces with -g:

```
# perf record -e bpf-output/no-inherit,name=evt/ -e ./kca_from.c/map:channel.event=evt/ -a -g -- sleep 1
bpf: builtin compilation failed: -95, try external compiler
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.215 MB perf.data (3 samples) ]

# perf script
testserver00001 16744 [002] 481518.262579:          0                evt:
                  410f51 kmem_cache_alloc (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9cb40f tcp_conn_request (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9da243 tcp_v4_conn_request (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9d0936 tcp_rcv_state_process (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9db102 tcp_v4_do_rcv (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9dcabf tcp_v4_rcv (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9b4af4 ip_local_deliver_finish (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9b4dff ip_local_deliver (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9b477b ip_rcv_finish (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9b50fb ip_rcv (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  97119e __netif_receive_skb_core (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  971708 __netif_receive_skb (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  9725df process_backlog (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  971c8e net_rx_action (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
                  a8e58d __do_softirq (/lib/modules/4.10.0-rc8-virtual/build/vmlinux)
```

1.3.9 火焰图

```
# git clone https://github.com/brendangregg/FlameGraph  # or download it from github
# cd FlameGraph
# perf record -F 99 -ag -- sleep 60
# perf script | ./stackcollapse-perf.pl > out.perf-folded
# cat out.perf-folded | ./flamegraph.pl > perf-kernel.svg
```



2 Ftrace(Functional Trace)

2.1 简介

Ftrace 是不仅仅是 Linux 内核函数跟踪，而是 Linux 内核整体跟踪框架。不仅支持静态、动态跟踪，更为重要的是支持时延跟踪（记录每个事件的时间）。

不过从最新的内核（4.9+）起，社区热衷于基于 eBPF 的整体跟踪框架。

2.2 安装

- 内核编译：
  可参考 Perf 中内核编译选项

- 开始使用：
  ftrace 使用 debugfs 来跟踪，因此事先需要挂载 debugfs，例如挂载到/debug 目录（或者其他任何目录）下：

```
- mkdir /debug
- mount -t debugfs nodev /debug
```

2.3 使用案例

2.3.1 基本功能

- Static tracing of block_rq_insert tracepoint

```
# cd /sys/kernel/debug/tracing
# echo 1 > events/block/block_rq_insert/enable
# cat trace_pipe
# echo 0 > events/block/block_rq_insert/enable
```

- Dynamic function tracing of tcp_retransmit_skb():

```
# cd /sys/kernel/debug/tracing
# echo tcp_retransmit_skb > set_ftrace_filter
# echo function > current_tracer
# cat trace_pipe
# echo nop > current_tracer
# echo > set_ftrace_filter
```

- Available tracing capabilities:

```
# cat available_tracers
blk function_graph mmiotrace wakeup_rt wakeup function nop
```

- What would a sysadmin do?

```
# cd /sys/kernel/debug/tracing
# echo tcp_retransmit_skb > set_ftrace_filter
# echo function > current_tracer
# cat trace_pipe
# echo nop > current_tracer
# echo > set_ftrace_filter
```

- Automate:

```
# functrace tcp_retransmit_skb
```

2.3.2 trace-cmd

可以借助 trace-cmd（需要单独安装）简化 ftrace 上述跟踪命令。

- Default, writes to “trace.dat”

```
[root@frodo ~]# trace-cmd record -e sched ls -ltr /usr > /dev/null
disable all
enable sched
offset=2f2000
offset=2f4000
[root@frodo ~]# trace-cmd record -o func.dat -p function ls -ltr /usr > /dev/null
  plugin function
disable all
offset=2f2000
offset=412000
[root@frodo ~]# trace-cmd record -o fgraph.dat -p function_graph ls -ltr /usr \
   > /dev/null
  plugin function_graph
disable all
offset=2f2000
offset=460000
[root@frodo ~]# trace-cmd record -o fgraph-events.dat -e sched -p function_graph \
    ls -ltr /usr > /dev/null
  plugin function_graph
disable all
enable sched
offset=2f2000
offset=461000
```

```
[root@frodo ~]# trace-cmd record -e sched_switch -f 'prev_prio < 100'

[root@frodo ~]# trace-cmd record -p function_graph -O nograph-time

[root@frodo ~]# trace-cmd record -p function_graph -g sys_read

[root@frodo ~]# trace-cmd record -p function_graph -l do_IRQ -l timer_interrupt

[root@frodo ~]# trace-cmd record -p function_graph -n '*lock*'
```

- -f : filter
- -O : option
- -g : same as echoing into set_graph_function
- -l : same as echoing into set_ftrace_filter
- -n : same as echoing into set_ftrace_notrace

```
[root@frodo ~]# trace-cmd report | head -15
version = 6
cpus=2
       trace-cmd-6157  [000]    83.713584: sched_stat_runtime:   task: trace-cmd:61
       trace-cmd-6157  [000]    83.713591: sched_switch:         6157:120:S ==> 0:1
          <idle>-0     [000]    83.713646: sched_stat_wait:      task: trace-cmd:61
          <idle>-0     [000]    83.713648: sched_switch:         0:120:R ==> 6158:1
              ls-6158  [001]    83.713934: sched_wakeup:         6158:?:? +   5900:
              ls-6158  [001]    83.713935: sched_stat_runtime:   task: trace-cmd:61
              ls-6158  [001]    83.713937: sched_stat_runtime:   task: trace-cmd:61
              ls-6158  [001]    83.713938: sched_switch:         6158:120:R ==> 590
     migration/1-5900  [001]    83.713941: sched_stat_wait:      task: trace-cmd:61
     migration/1-5900  [001]    83.713942: sched_migrate_task:   task trace-cmd:615
     migration/1-5900  [001]    83.713947: sched_switch:         5900:0:S ==> 0:120
              ls-6158  [000]    83.714067: sched_stat_runtime:   task: ls:6158 runt
              ls-6158  [000]    83.714636: sched_stat_runtime:   task: ls:6158 runt
```

- Using start is like echoing into debugfs
  - trace-cmd start -e all
    - same as “echo 1 > events/enable”
- Uses the same options as trace-cmd record
  - `trace-cmd start -p function_graph`
  - `trace-cmd start -p function -e sched_switch`

- trace-cmd stop
  - stops the tracer from writing:
    - same as “echo 0 > tracing_on”
- trace-cmd extract -o output.dat
  - Makes a “dat” file that trace-cmd report can use
  - Without “-o ...” will create “trace.dat”

- trace-cmd reset disables all tracing
  - `trace-cmd reset`

- trace-cmd list -o
  - shows list of trace options
  - these options are used by trace-cmd record -O option
- trace-cmd list -p
  - available plugins
- trace-cmd list -e
  - available events

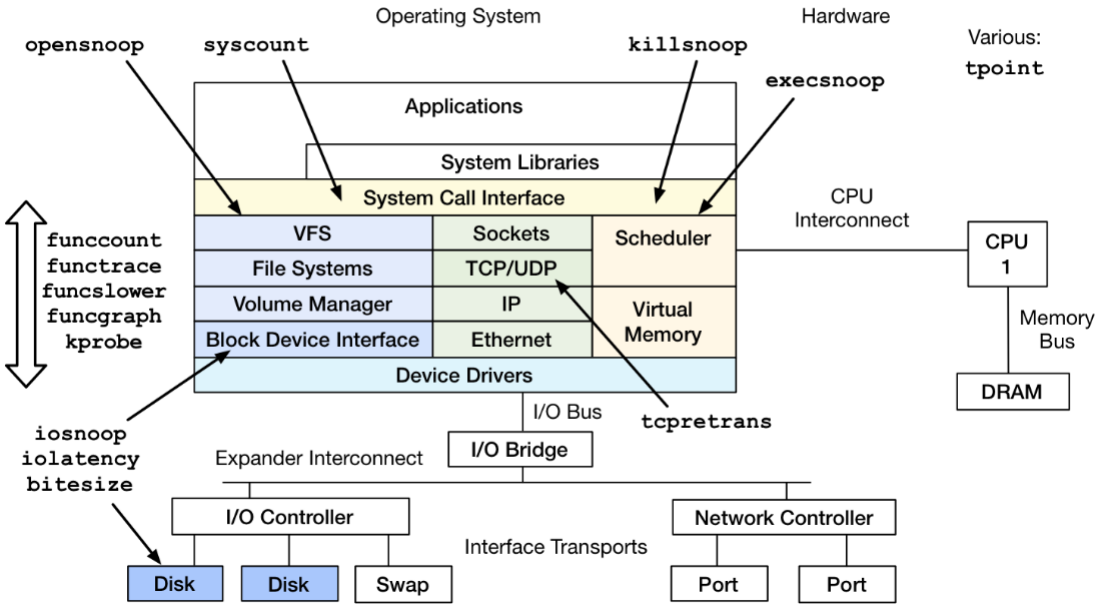
- trace-cmd split 258.121328
  - splits from timestamp to end of file
- trace-cmd split -e 1000
  - splits out the first 1000 events
- trace-cmd split -m 1 -r 258.121328 259.000000
  - split 1 millisecond starting at first timestamp to second timestamp repeatedly
    - trace.dat.1, trace.dat.2, ...

- listen for connections from other boxes
  - `trace-cmd listen -p 5678 -d`
- Record can now send to that box
  - `trace-cmd record -N host:5678 -e all`
  - use "-t" to force TCP otherwise trace data is sent via UDP

2.3.3 Ftrace-Tools

- A collection of tools for both ftrace and perf_events
  - https://github.com/brendangregg/perf-tools

perf-tools (so far…)



Tool	Description
iosnoop	trace disk I/O with details including latency
iolatency	summarize disk I/O latency as a histogram
execsnoop	trace process exec() with command line argument details
opensnoop	trace open() syscalls showing filenames
killsnoop	trace kill() signals showing process and signal details
syscount	count syscalls by syscall or process
disk/bitesize	histogram summary of disk I/O size
net/tcpretrans	show TCP retransmits, with address and other details
tools/reset-ftrace	reset ftrace state if needed
system/tpoint	trace a given tracepoint
kernel/funccount	count kernel function calls, matching a string
kernel/functrace	trace kernel function calls, matching a string
kernel/funcslower	trace kernel functions slower than a threshold
kernel/funcgraph	graph kernel function calls, showing children and times
kernel/kprobe	dynamically trace a kernel function call or its return, with variables

3 BPF/eBPF/BCC

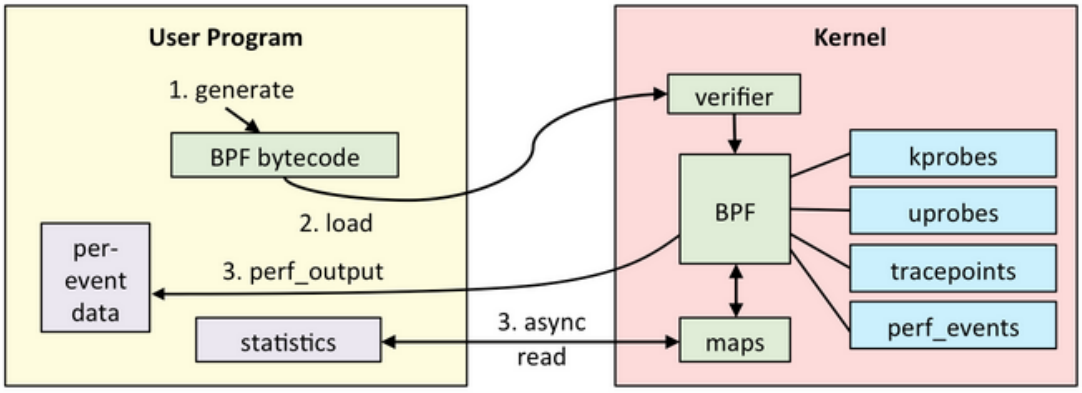
3.1 简介

BPF 早期仅用来处理网络报文过滤的，而 eBPF(Extended BFP)（不能仅仅看做做为 BPF 的扩展，因为两者之间有较大差异）衍生为内核整体跟踪框架。与 perf/ftrace 相比，更为重要的是更为安全，对生产环境造成的影响更小，因为其 BPF 代码是在内核虚拟机上动态编译运行的。

- Extended BPF: programs on tracepoints
  - High performance filtering: JIT
  - In-kernel summaries: maps

不过，直接编写 BPF（类似汇编代码）比较困难，因此一般借助第三方工具（例如 bcc）。

3.2 总体框架



3.3 使用案例

以下工具都是基于 bcc 的。

3.3.1 基本功能

```
# Trace new processes:
execsnoop

# Trace file opens with process and filename:
opensnoop

# Summarize block I/O (disk) latency as a power-of-2 distribution by disk:
biolatency -D

# Summarize block I/O size as a power-of-2 distribution by program name:
bitesize

# Trace common ext4 file system operations slower than 1 millisecond:
ext4slower 1

# Trace TCP active connections (connect()) with IP address and ports:
tcpconnect

# Trace TCP passive connections (accept()) with IP address and ports:
tcpaccept

# Trace TCP connections to local port 80, with session duration:
tcplife -L 80

# Trace TCP retransmissions with IP addresses and TCP state:
tcpretrans

# Sample stack traces at 49 Hertz for 10 seconds, emit folded format (for flame graphs):
profile -fd -F 49 10

# Trace details and latency of resolver DNS lookups:
gethostlatency

# Trace commands issued in all running bash shells:
bashreadline
```

3.3.2 内核动态跟踪

```
# Count "tcp_send*" kernel function, print output every second:
funccount -i 1 'tcp_send*'

# Count "vfs_*" calls for PID 185:
funccount -p 185 'vfs_*'

# Trace file names opened, using dynamic tracing of the kernel do_sys_open() function:
trace 'p::do_sys_open "%s", arg2'

# Same as before ("p:: is assumed if not specified):
trace 'do_sys_open "%s", arg2'

# Trace the return of the kernel do_sys_open() funciton, and print the retval:
trace 'r::do_sys_open "ret: %d", retval'

# Trace do_nanosleep() kernel function and the second argument (mode), with kernel stack traces:
trace -K 'do_nanosleep "mode: %d", arg2'

# Trace do_nanosleep() mode by providing the prototype (no debuginfo required):
trace 'do_nanosleep(struct hrtimer_sleeper *t, enum hrtimer_mode mode) "mode: %d", mode'

# Trace do_nanosleep() with the task address (may be NULL), noting the dereference:
trace 'do_nanosleep(struct hrtimer_sleeper *t, enum hrtimer_mode mode) "task: %x", t->task'

# Frequency count tcp_sendmsg() size:
argdist -C 'p::tcp_sendmsg(struct sock *sk, struct msghdr *msg, size_t size):u32:size'

# Summarize tcp_sendmsg() size as a power-of-2 histogram:
argdist -H 'p::tcp_sendmsg(struct sock *sk, struct msghdr *msg, size_t size):u32:size'

# Frequency count stack traces that lead to the submit_bio() function (disk I/O issue):
stackcount submit_bio

# Summarize the latency (time taken) by the vfs_read() function for PID 181:
funclatency -p 181 -u vfs_read
```

3.3.3 用户动态跟踪

```
# Trace the libc library function nanosleep() and print the requested sleep details:
trace 'p:c:nanosleep(struct timespec *req) "%d sec %d nsec", req->tv_sec, req->tv_nsec'

# Count the libc write() call for PID 181 by file descriptor:
argdist -p 181 -C 'p:c:write(int fd):int:fd'

# Summarize the latency (time taken) by libc getaddrinfo(), as a power-of-2 histogram in microseconds:
funclatency.py -u 'c:getaddrinfo'
```

3.3.4 内核静态跟踪

```
# Count stack traces that led to issuing block I/O, tracing its kernel tracepoint:
stackcount t:block:block_rq_insert
```

3.3.5 用户静态跟踪

```
# Trace the pthread_create USDT probe, and print arg1 as hex:
trace 'u:pthread:pthread_create "%x", arg1'
```

3.3.6 性能检查列表

1. execsnoop

Trace new processes via exec() syscalls, and print the parent process name and other details:

```
# execsnoop
PCOMM            PID    RET ARGS
bash             15887    0 /usr/bin/man ls
preconv          15894    0 /usr/bin/preconv -e UTF-8
man              15896    0 /usr/bin/tbl
man              15897    0 /usr/bin/nroff -mandoc -rLL=169n -rLT=169n -Tutf8
man              15898    0 /usr/bin/pager -s
nroff            15900    0 /usr/bin/locale charmap
nroff            15901    0 /usr/bin/groff -mtty-char -Tutf8 -mandoc -rLL=169n -rLT=169n
groff            15902    0 /usr/bin/troff -mtty-char -mandoc -rLL=169n -rLT=169n -Tutf8
groff            15903    0 /usr/bin/grotty
[...]
```

2. opensnoop

Trace open() syscalls and print process name and path name details:

```
# opensnoop
PID    COMM               FD ERR PATH
27159  catalina.sh         3   0 /apps/tomcat8/bin/setclasspath.sh
4057   redis-server        5   0 /proc/4057/stat
2360   redis-server        5   0 /proc/2360/stat
30668  sshd                4   0 /proc/sys/kernel/ngroups_max
30668  sshd                4   0 /etc/group
30668  sshd                4   0 /root/.ssh/authorized_keys
30668  sshd                4   0 /root/.ssh/authorized_keys
30668  sshd               -1   2 /var/run/nologin
30668  sshd               -1   2 /etc/nologin
30668  sshd                4   0 /etc/login.defs
30668  sshd                4   0 /etc/passwd
30668  sshd                4   0 /etc/shadow
30668  sshd                4   0 /etc/localtime
4510   snmp-pass           4   0 /proc/cpuinfo
[...]
```

3. ext4slower

Trace slow ext4 operations that are slower than a provided threshold (bcc has versions of this for btrfs, XFS, and ZFS as well):

```
# ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:49:17 bash           3616   R 128     0           7.75 cksum
06:49:17 cksum          3616   R 39552   0           1.34 [
06:49:17 cksum          3616   R 96      0           5.36 2to3-2.7
06:49:17 cksum          3616   R 96      0          14.94 2to3-3.4
06:49:17 cksum          3616   R 10320   0           6.82 411toppm
06:49:17 cksum          3616   R 65536   0           4.01 a2p
06:49:17 cksum          3616   R 55400   0           8.77 ab
06:49:17 cksum          3616   R 36792   0          16.34 aclocal-1.14
06:49:17 cksum          3616   R 15008   0          19.31 acpi_listen
06:49:17 cksum          3616   R 6123    0          17.23 add-apt-repository
06:49:17 cksum          3616   R 6280    0          18.40 addpart
06:49:17 cksum          3616   R 27696   0           2.16 addr2line
06:49:17 cksum          3616   R 58080   0          10.11 ag
```

4. biolatency

Summarize block device I/O latency as a histogram every second:

```
# biolatency -mT 1
Tracing block device I/O... Hit Ctrl-C to end.

21:33:40
     msecs               : count     distribution
         0 -> 1          : 69       |****************************************|
         2 -> 3          : 16       |*********                               |
         4 -> 7          : 6        |***                                     |
         8 -> 15         : 21       |************                            |
        16 -> 31         : 16       |*********                               |
        32 -> 63         : 5        |**                                      |
        64 -> 127        : 1        |                                        |
```

5. biosnoop

Trace block device I/O with process, disk, and latency details:

```
# biosnoop
TIME(s)        COMM           PID    DISK    T  SECTOR    BYTES   LAT(ms)
0.000004001    supervise      1950   xvda1   W  13092560  4096       0.74
0.000178002    supervise      1950   xvda1   W  13092432  4096       0.61
0.001469001    supervise      1956   xvda1   W  13092440  4096       1.24
0.001588002    supervise      1956   xvda1   W  13115128  4096       1.09
1.022346001    supervise      1950   xvda1   W  13115272  4096       0.98
1.022568002    supervise      1950   xvda1   W  13188496  4096       0.93
1.023534000    supervise      1956   xvda1   W  13188520  4096       0.79
1.023585003    supervise      1956   xvda1   W  13189512  4096       0.60
```

6. cachestat

Show the page cache hit/miss ratio and size, and summarize every second:

```
# cachestat
    HITS   MISSES  DIRTIES  READ_HIT% WRITE_HIT%   BUFFERS_MB  CACHED_MB
  170610    41607       33      80.4%      19.6%           11        288
  157693     6149       33      96.2%       3.7%           11        311
  174483    20166       26      89.6%      10.4%           12        389
  434778       35       40     100.0%       0.0%           12        389
  435723       28       36     100.0%       0.0%           12        389
  846183    83800   332534      55.2%       4.5%           13        553
   96387       21       24     100.0%       0.0%           13        553
  120258       29       44      99.9%       0.0%           13        553
  255861       24       33     100.0%       0.0%           13        553
  191388       22       32     100.0%       0.0%           13        553
[...]
```

7. tcpconnect

Trace TCP active connections (connect()):

```
# tcpconnect
PID    COMM         IP SADDR            DADDR            DPORT
25333  recordProgra 4  127.0.0.1        127.0.0.1        28527
25338  curl         4  100.66.3.172     52.22.109.254    80
25340  curl         4  100.66.3.172     31.13.73.36      80
25342  curl         4  100.66.3.172     104.20.25.153    80
25344  curl         4  100.66.3.172     50.56.53.173     80
25365  recordProgra 4  127.0.0.1        127.0.0.1        28527
```

8. tcpaccept

Trace TCP passive connections (accept()):

```
# tcpaccept
PID    COMM         IP RADDR            LADDR            LPORT
2287   sshd         4  11.16.213.254    100.66.3.172     22
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
2287   sshd         6  ::1              ::1              22
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
2287   sshd         6  fe80::8a3:9dff:fed5:6b19 fe80::8a3:9dff:fed5:6b19 22
4057   redis-server 4  127.0.0.1        127.0.0.1        28527
[...]
```

9. tcpretrans

Trace TCP retransmits and TLPs:

```
# tcpretrans
TIME     PID    IP LADDR:LPORT          T> RADDR:RPORT          STATE
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:17 0      4  10.153.223.157:22    R> 69.53.245.40:22957   ESTABLISHED
[...]
```

10. gethostlatency

Show latency for getaddrinfo/gethostbyname[2] library calls, system wide:

```
# gethostlatency
TIME      PID    COMM           LATms HOST
06:10:24  28011  wget           90.00 www.iovisor.org
06:10:28  28127  wget            0.00 www.iovisor.org
06:10:41  28404  wget            9.00 www.netflix.com
06:10:48  28544  curl           35.00 www.netflix.com.au
06:11:10  29054  curl           31.00 www.plumgrid.com
06:11:16  29195  curl            3.00 www.facebook.com
06:11:24  25313  wget            3.00 www.usenix.org
06:11:25  29404  curl           72.00 foo
06:11:28  29475  curl            1.00 foo
[...]
```

11. runqlat

Show run queue (scheduler) latency as a histogram, every 5 seconds:

```
# runqlat -m 5
Tracing run queue latency... Hit Ctrl-C to end.

     msecs               : count     distribution
         0 -> 1          : 2085     |****************************************|
         2 -> 3          : 8        |                                        |
         4 -> 7          : 20       |                                        |
         8 -> 15         : 191      |***                                     |
        16 -> 31         : 420      |********                                |

     msecs               : count     distribution
         0 -> 1          : 1798     |****************************************|
         2 -> 3          : 11       |                                        |
         4 -> 7          : 45       |*                                       |
         8 -> 15         : 441      |*********                               |
        16 -> 31         : 1030     |**********************                  |
```

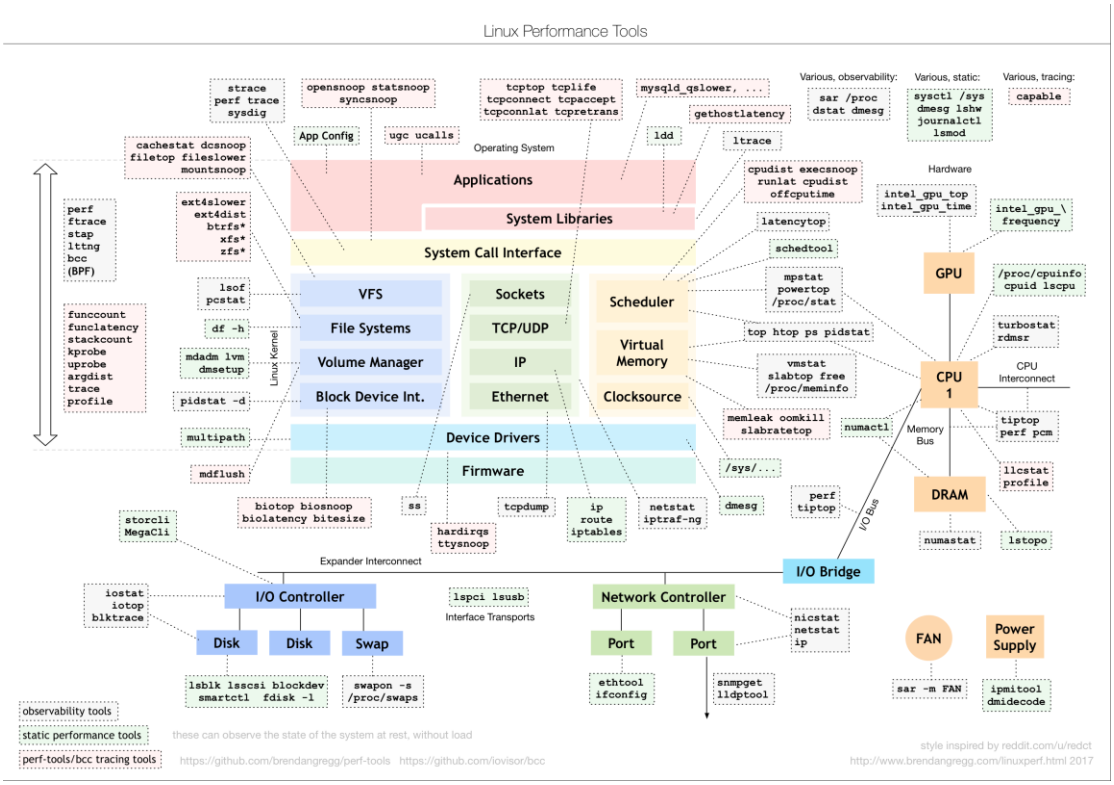
12. profile

Sample stack traces at 49 Hertz, then print unique stacks with the number of occurrences seen:

```
# profile
Sampling at 49 Hertz of all threads by user + kernel stack... Hit Ctrl-C to end.
^C
[...]
    ffffffff811a2eb0 find_get_entry
    ffffffff811a338d pagecache_get_page
    ffffffff811a51fa generic_file_read_iter
    ffffffff81231f30 __vfs_read
    ffffffff81233063 vfs_read
    ffffffff81234565 SyS_read
    ffffffff818739bb entry_SYSCALL_64_fastpath
    00007f4757ff9680 read
    -                dd (14283)
        29

    ffffffff8141c067 copy_page_to_iter
    ffffffff811a54e8 generic_file_read_iter
    ffffffff81231f30 __vfs_read
    ffffffff81233063 vfs_read
    ffffffff81234565 SyS_read
    ffffffff818739bb entry_SYSCALL_64_fastpath
    00007f407617d680 read
    -                dd (14288)
        32
```

4 总结



5 参考文献

[1] https://www.kernel.org/
[2] http://www.brendangregg.com
[3] https://github.com/iovisor/bcc