Ferramentas de Observabilidade

A Comunidade "perf" no Kernel Linux

Arnaldo Carvalho de Melo

Red Hat Inc.

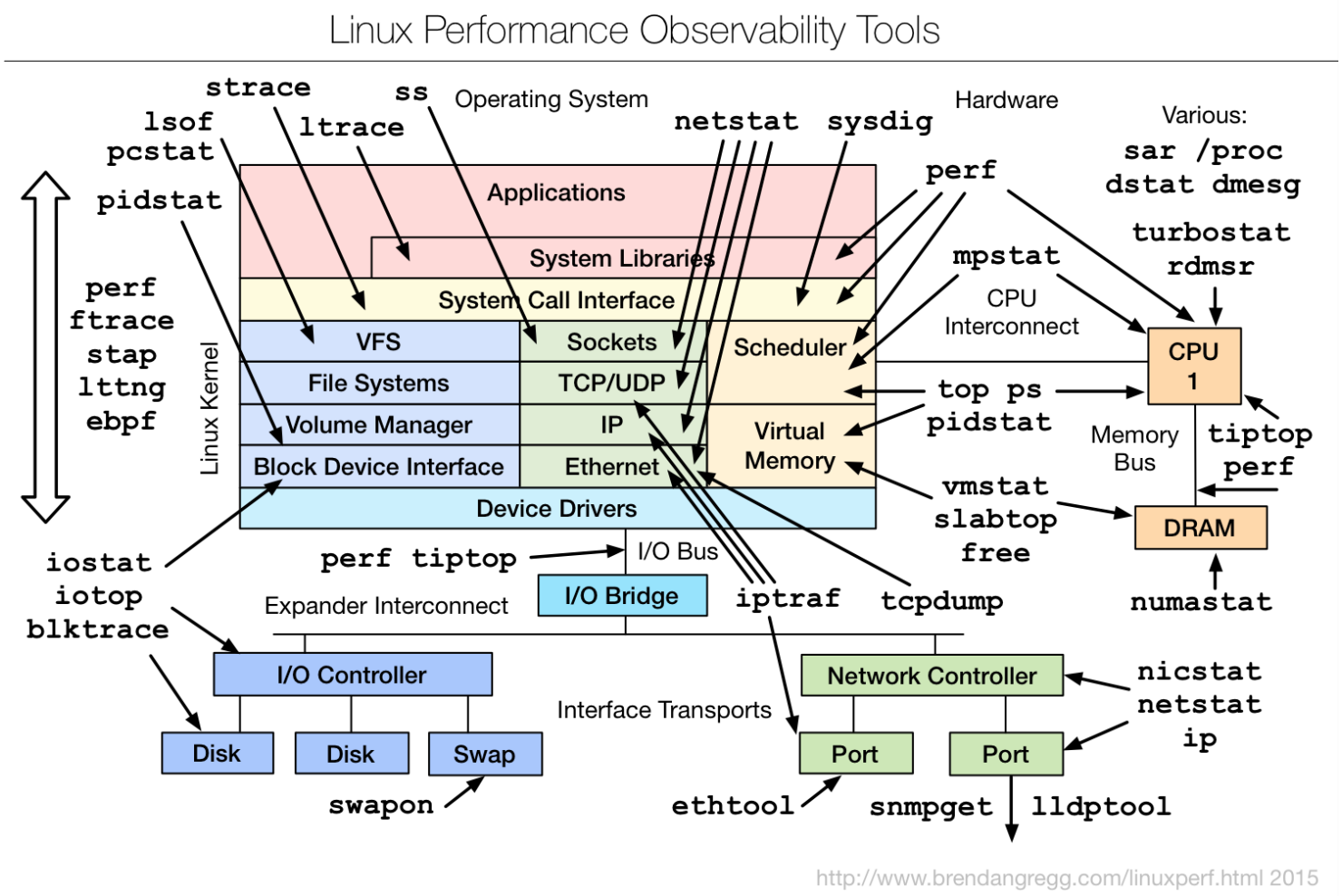
October 15, 2015

Motivação

- Complexidade crescente do hardware e software
- Linux em todos estes cenários:
- Real Time/NFV
- Virtualização/Cloud
- Smartphones/Android/Embedded
- IoT

`http://vger.kernel.org/~acme/perf/cinfotec2015.pdf`

Ferramentas de Observabilidade no Linux



http://www.brendangregg.com/linuxperf.html

Ferramentas de Observabilidade

- Custo de observação:
- Alteração do comportamento do que é observado
- Flexibilidade

Overview

- Alternativas às ferramentas existentes
- Mas com novas funcionalidades
- Usando infraestruturas (relativamente) recentes do kernel
- perf, ftrace, kprobes, uprobes
- Mais: eBPF, Intel PT

Overview

- Exemplos do funcionamento das ferramentas
- Sem requerer 'root' tanto quanto possível
- Mencionando público alvo das funcionalidades
- Desenvolvedores, estudantes, administradores de sistemas, etc

Comunidade do Kernel: 4.1 ao v4.3-rc4

1	2802	Intel
2	2174	Red Hat
3	903	Samsung
4	739	Linaro
5	722	SuSE
6	523	IBM
7	512	AMD
8	471	TI
9	405	Freescale
10	390	Renesas

Comunidade do tools/perf: 4.1 ao v4.3-rc4

```
  1      312 Red Hat
  2      190 Intel
  3       77 Huawei
  4       50 LG
  5       40 Hitachi
  6       17 IBM
  7        9 Google
  8        5 Unknown
  9        5 SuSE
 10        5 Oracle
```

Contribuição recente: Intel

- Intel PT - Processor trace
- Microarquitetura Broadwell
- Tracing de branchs, calls, jumps diretamente pela CPU
- Grande volume de dados
- Suporte a outras funcionalidades em novos processadores

Contribuição recente: Google

- Análise de código JIT
- Java, PHP, etc
- Coordena com JVM para obter tabelas de símbolos
- Gera arquivos ELF para permitir anotação de código

Contribuição recente: Huawei

- Integração com eBPF
- VM no kernel para agregação de informações
- Redução de volume de amostragem
- Scripts em C, usando clang/LLVM
- Geração de código de máquina nativo
- Agregação/filtragem na origem, no kernel
- Interesse da Huawei: Depuração avançada em Smartphones
- Exemplo recente: O que ocorre em uma atualização de tela?

perf stat

```
[acme@zoo cinfotec2015]$ perf stat make

<saída do comando make foi suprimida>

 Performance counter stats for 'make':

   1646.724682 task-clock (msec) # 0.994 CPUs utilized
         1,080 context-switches  # 0.656 K/sec
            18 cpu-migrations    # 0.011 K/sec
         8,015 page-faults       # 0.005 M/sec
 4,952,541,444 cycles            # 3.008 GHz
 8,465,480,671 instructions      # 1.71 insns per cycle
 1,767,196,096 branches          # 1073.158 M/sec
    27,243,179 branch-misses     # 1.54% of all branches

   1.656747532 seconds time elapsed

[acme@zoo cinfotec2015]$
```

perf stat para todo o sistema

```
[root@zoo ~]# perf stat -a sleep 1

 Performance counter stats for 'system wide':

 4009.969839 task-clock (msec) # 4.004 CPUs utilized
        9104 context-switches  # 0.002 M/sec
         393 cpu-migrations    # 0.098 K/sec
         775 page-faults       # 0.193 K/sec
   540063094 cycles            # 0.135 GHz
   194430545 instructions      # 0.36  insns per cycle
    40587857 branches          # 10.122 M/sec
     2373582 branch-misses     # 5.85% of all branches

 1.001599059 seconds time elapsed

[root@zoo ~]#
```

Contando tracepoints

```
[root@zoo ~]# perf stat -a -e ext4:*_allocate_*,ext4:*_free_* sleep

 Performance counter stats for 'system wide':

    39  ext4:ext4_allocate_inode
     1  ext4:ext4_allocate_blocks
    27  ext4:ext4_free_inode
    15  ext4:ext4_free_blocks

       30.000751455 seconds time elapsed

[root@zoo ~]#
```

perf list sw

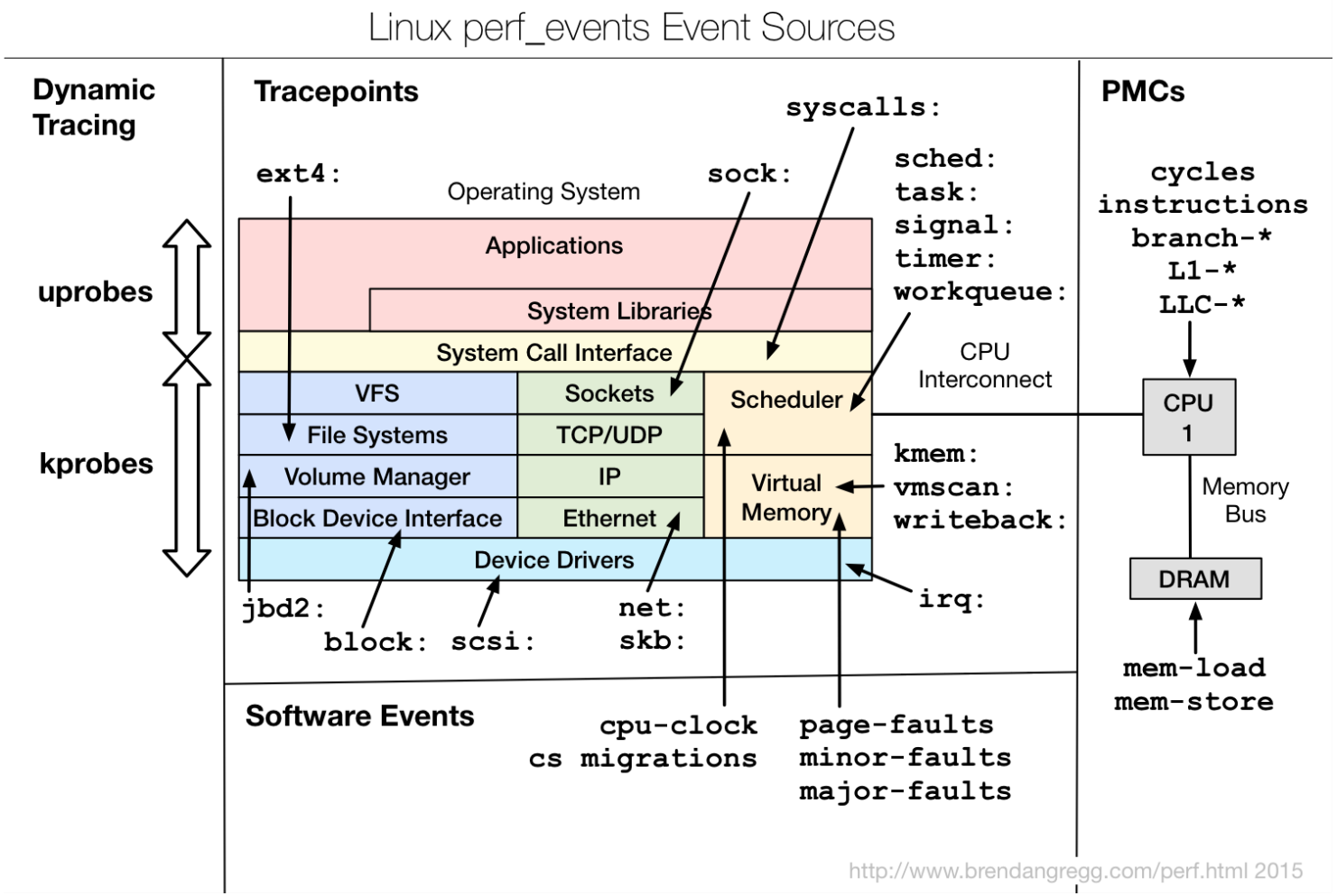
```
[acme@zoo ~]$ perf list sw

List of pre-defined events (to be used in -e):

  alignment-faults                   [Software event]
  context-switches OR cs             [Software event]
  cpu-clock                          [Software event]
  cpu-migrations OR migrations       [Software event]
  dummy                              [Software event]
  emulation-faults                   [Software event]
  major-faults                       [Software event]
  minor-faults                       [Software event]
  page-faults OR faults              [Software event]
  task-clock                         [Software event]

[acme@zoo ~]$
```

perf list

- 67 subsistemas com tracepoints
- Subsistemas com mais tracepoints

```
600   syscalls
141   cfg80211
111   mac80211
 95   ext4
 50   kvm
 36   i915
 35   xen
 28   writeback
 26   sunrpc
 23   sched
```

Mapa de Eventos



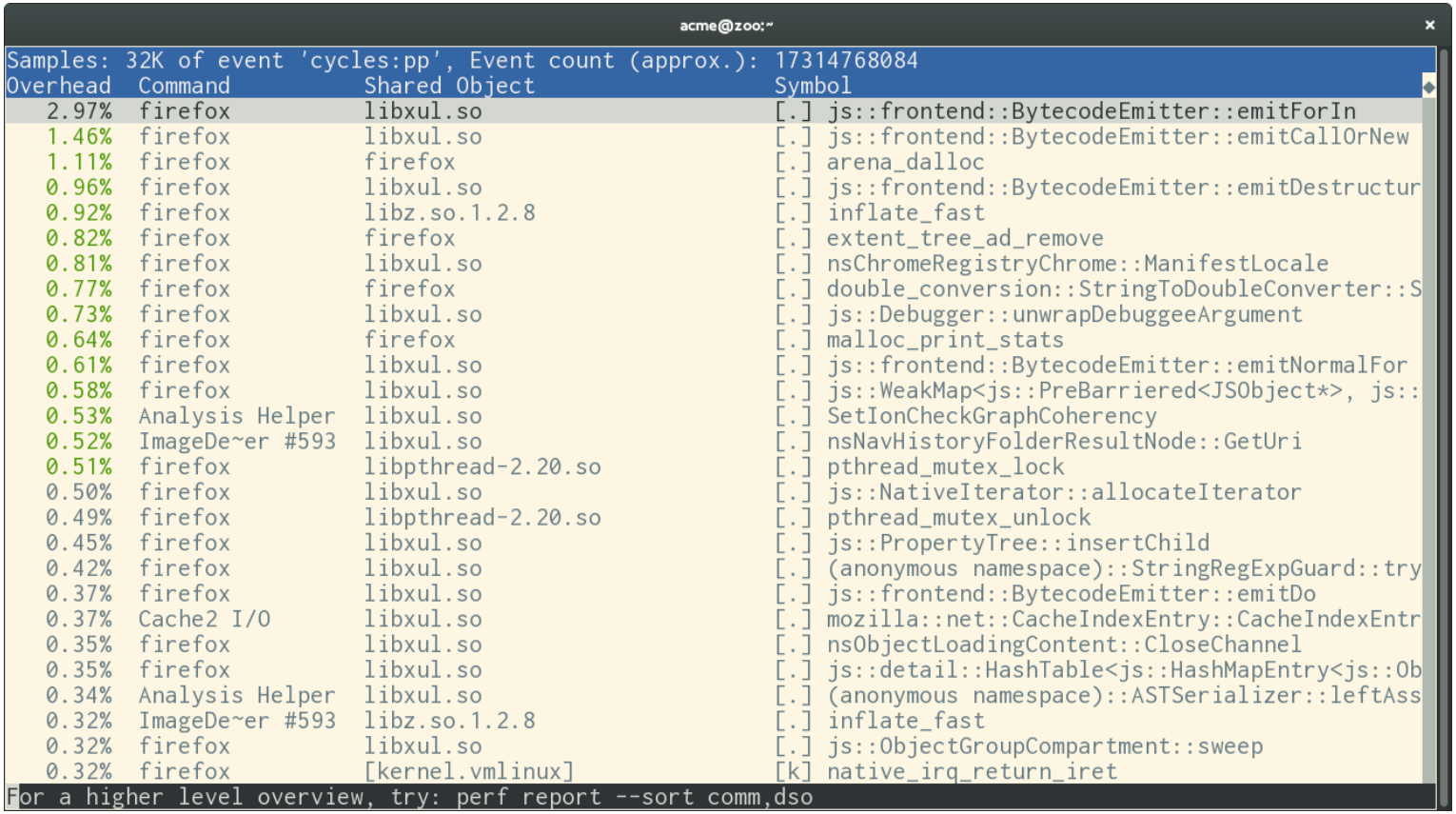
http://www.brendangregg.com/perf.html

perf record

```
$ pidof firefox
20739
$ perf record -p 20739 sleep 30
[ perf record: Woken up 4 times to write data ]
[ perf record: Captured and wrote 2.776 MB perf.data (32850 samples)
$ ls -lah perf.data
-rw-------. 1 acme acme 2.8M Oct 13 12:38 perf.data
$
```

perf report

```
# perf report
```



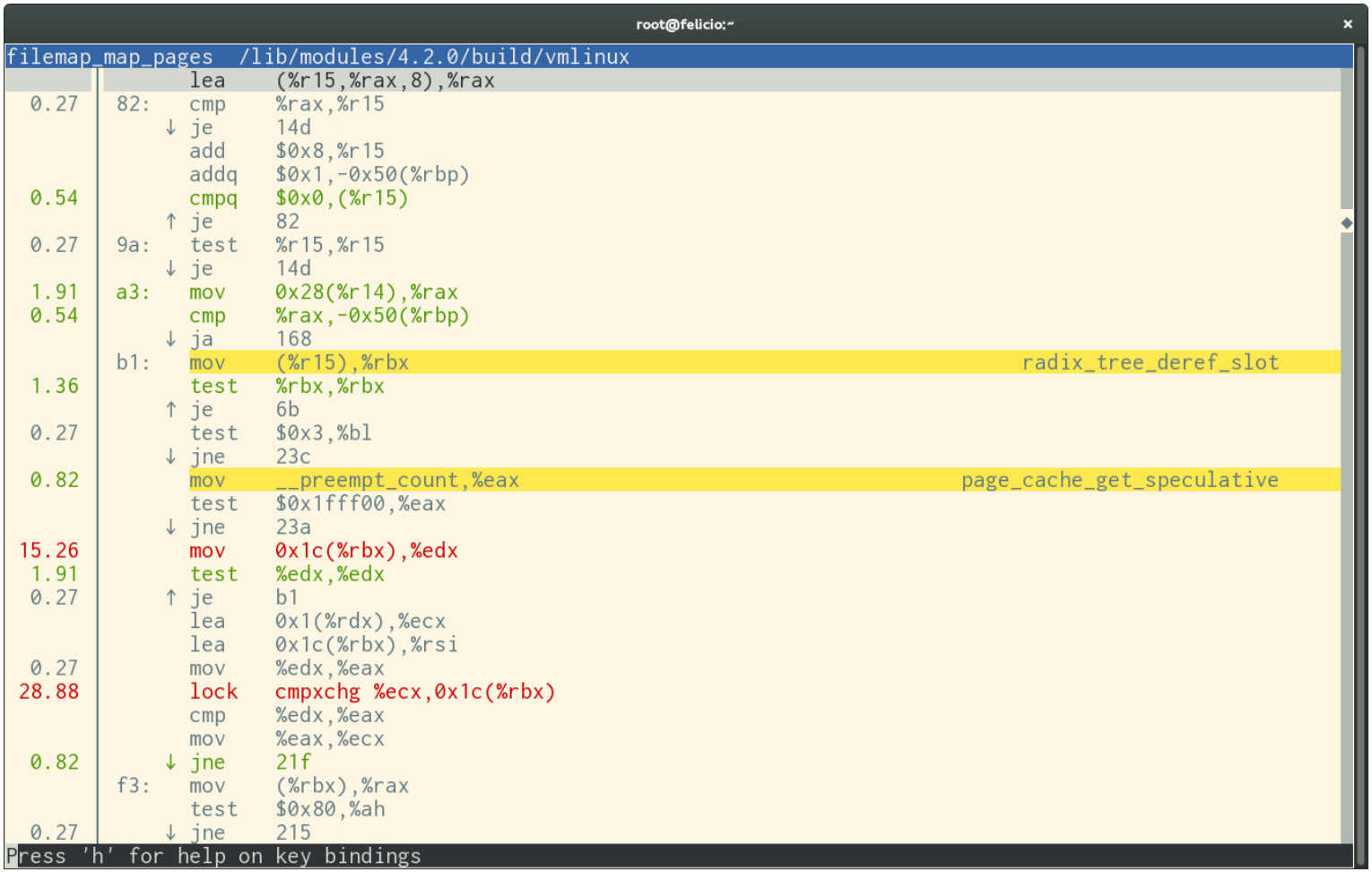
perf report –sort comm,dso

```
# perf report --sort comm,dso
```

acme@zoo:~

```
Samples: 32K of event 'cycles:pp', Event count (approx.): 17314768084
Overhead  Command          Shared Object
  60.66%  firefox          libxul.so
   6.52%  Analysis Helper  libxul.so
   5.53%  firefox          [kernel.vmlinux]
   4.71%  firefox          firefox
   2.41%  firefox          perf-20739.map
   1.38%  firefox          libpthread-2.20.so
   1.19%  firefox          libfontconfig.so.1.8.0
   1.19%  Cache2 I/O       libxul.so
   1.11%  DOM Worker       libxul.so
   1.07%  firefox          libz.so.1.2.8
   0.95%  firefox          libc-2.20.so
   0.83%  Analysis Helper  [kernel.vmlinux]
   0.77%  ImageDe~er #593  libxul.so
   0.70%  firefox          libglib-2.0.so.0.4200.2
   0.60%  Cache2 I/O       [kernel.vmlinux]
   0.51%  Socket Thread    p11-kit-trust.so
   0.46%  Socket Thread    libfreebl3.so
   0.46%  firefox          libgobject-2.0.so.0.4200.2
   0.43%  Socket Thread    firefox
   0.42%  firefox          libgdk-x11-2.0.so.0.2400.28
   0.41%  Analysis Helper  firefox
   0.41%  ImageDe~er #593  libz.so.1.2.8
   0.37%  Timer            [kernel.vmlinux]
   0.36%  Socket Thread    libxul.so
   0.35%  DOM Worker       [kernel.vmlinux]
   0.33%  firefox          libX11.so.6.3.0
For a higher level overview, try: perf report --sort comm,dso
```

Annotate


root@felicio:~
filemap_map_pages /lib/modules/4.2.0/build/vmlinux
lea (%r15,%rax,8),%rax
0.27 82: cmp %rax,%r15
↓ je 14d
add $0x8,%r15
addq $0x1,-0x50(%rbp)
0.54 cmpq $0x0,(%r15)
↑ je 82
0.27 9a: test %r15,%r15
↓ je 14d
1.91 a3: mov 0x28(%r14),%rax
0.54 cmp %rax,-0x50(%rbp)
↓ ja 168
b1: mov (%r15),%rbx radix_tree_deref_slot
1.36 test %rbx,%rbx
↑ je 6b
0.27 test $0x3,%bl
↓ jne 23c
0.82 mov __preempt_count,%eax page_cache_get_speculative
test $0x1fff00,%eax
↓ jne 23a
15.26 mov 0x1c(%rbx),%edx
1.91 test %edx,%edx
0.27 ↑ je b1
lea 0x1(%rdx),%ecx
lea 0x1c(%rbx),%rsi
0.27 mov %edx,%eax
28.88 lock cmpxchg %ecx,0x1c(%rbx)
cmp %edx,%eax
mov %eax,%ecx
0.82 ↓ jne 21f
f3: mov (%rbx),%rax
test $0x80,%ah
0.27 ↓ jne 215
Press 'h' for help on key bindings

trace

- strace
- Não utiliza a infraestrutura ptrace: menor custo
- Não para o programa sendo observado
- Decorador de argumentos de syscalls
- Mais alvos: todo o sistema, cpu(s), cgroups, outros
- Mesmo workflow do record/report

Custo do strace

```
# dd if=/dev/zero of=/dev/null bs=1 count=500k
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.148031 s, 3.5 MB/s
#
```

Custo do strace

```
# strace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 11.6152 s, 44.1 kB/s
```

- 78x mais lento
- Exemplo do pior caso, pois o dd emite syscalls tão rápido quanto pode
- Artigo do Brendan Gregg cita 442x mais lento, mesmo exemplo
- ptrace pausa a aplicação duas vezes
- Uma vez na entrada da syscall e outra na saída
- Para em cada syscall, mesmo com somente a 'accept' selecionada
- www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html

Custo do perf trace

```
perf trace -eaccept dd if=/dev/zero of=/dev/null bs=1 count=500k
512000+0 records in
512000+0 records out
512000 bytes (512 kB) copied, 0.300745 s, 1.7 MB/s
```

- 2x mais lento
- Pior caso novamente, o 'dd' emite syscalls tão rápido quanto possível
- Programa observado não é interrompido na entrada e na saída da syscall
- Coleta informação sem troca de contexto para a o programa monitor
- Mais otimizações são possíveis
- Atualmente usa os tracepoints raw_syscalls:sys_{enter,exit}
- É notificado para cada syscall, mesmo com somente 'accept' selecionada
- Mas neste exemplo poderia ter usado apenas syscalls:sys_{enter,exit}_accept

Permissões

- Múltiplos mecanismos do kernel são utilizados
- Vários ajustes são possíveis no debugfs, tracepoints
- Configurações no /proc

```
# ls /proc/sys/kernel/perf_event_*
/proc/sys/kernel/perf_event_max_sample_rate
/proc/sys/kernel/perf_event_mlock_kb
/proc/sys/kernel/perf_event_paranoid
# cat /proc/sys/kernel/perf_event_max_sample_rate
50000
# cat /proc/sys/kernel/perf_event_mlock_kb
516
# cat /proc/sys/kernel/perf_event_paranoid
1
```

perf_event_paranoid

```
$ perf top --stdio
Error:
You may not have permission to collect system-wide stats.
Consider tweaking /proc/sys/kernel/perf_event_paranoid:
 -1 - Not paranoid at all
  0 - Disallow raw tracepoint access for unpriv
  1 - Disallow cpu events for unpriv
  2 - Disallow kernel profiling for unpriv
$
```

Permissões

```
$ ps ax|grep git
 3328 pts/0    S+     0:00 git remote update clark
 3329 pts/0    S+     0:00 git fetch --multiple clark
 3330 pts/0    S+     0:04 git fetch --append clark
 3370 pts/1    S+     0:00 grep git
$ trace -p 3330
Error: Unable to find debugfs
Hint: Was your kernel compiled with debugfs support?
Hint: Is the debugfs filesystem mounted?
Hint: Try 'sudo mount -t debugfs nodev /sys/kernel/debug'
```

Permissions

```
$ sudo mount -t debugfs nodev /sys/kernel/debug
$ trace -p 3330
Error: No permissions to read /sys/kernel/debug/tracing/events/raw_syscalls/*
Hint:  Try 'sudo mount -o remount,mode=755 /sys/kernel/debug'
```

Permissões

```
$ sudo mount -o remount,mode=755 /sys/kernel/debug
$ trace -p 3330
   0.012 (  0.006 ms): read(fd: 30<socket:[849]>, buf: 0x7215a0, count: 4      ) = 4
   0.070 (  0.017 ms): write(fd: 31<socket:[849]>, buf: 0x11e12c0, count: 1604 ) = 1604
 267.908 (267.836 ms): read(fd: 30<socket:[849]>, buf: 0x7fff503efb20, count: 4) = 4
 267.914 (  0.004 ms): read(fd: 30<socket:[849]>, buf: 0x7215a0, count: 4      ) = 4
 267.945 (  0.011 ms): write(fd: 31<socket:[849]>, buf: 0x11e12c0, count: 1604 ) = 1604
 533.696 (265.749 ms): read(fd: 30<socket:[849]>, buf: 0x7fff503efb20, count: 4) = 4
 533.701 (  0.003 ms): read(fd: 30<socket:[849]>, buf: 0x7215a0, count: 4      ) = 4
 533.800 (  0.013 ms): write(fd: 31<socket:[849]>, buf: 0x11e12c0, count: 1604 ) = 1604
^C
#
```

Tracing de todo o sistema sem root

```
$ trace
Error: Operation not permitted.
Hint: Check /proc/sys/kernel/perf_event_paranoid setting.
Hint: For system wide tracing it needs to be set to -1.
Hint: Try: 'sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"'
Hint: The current value is 1.
$
```

Tracing de todo o sistema sem root

```
$ sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
$ trace  | head -10
 243.634 (0.000 ms): qpidd/382 ... [continued]: futex()) = -1 ETIMEDOUT Connection timed out
 243.654 (0.004 ms): qpidd/382 futex(uaddr: 0x7cb578, op: WAKE|PRIV, val: 1) = 0
2243.629 (1999.962 ms): qpidd/382 futex(uaddr: 0x7cb5a4, op: WAIT_BITSET|PRIV|CLKRT,
                                        val: 3775, utime: 0x7ffaa7a7bc90,
                                      val3: 4294967295) = -1 ETIMEDOUT Connection timed out
2243.636 (0.002 ms): qpidd/382 futex(uaddr: 0x7cb578,
                                     op: WAKE|PRIV, val: 1) = 0
2249.243 (0.000 ms): hald-addon-acp/875  ... [continued]: nanosleep()) = 0
2249.401 (0.154 ms): hald-addon-acp/875 socket(family: LOCAL,
                                               type: STREAM) = 4
2249.424 (0.020 ms): hald-addon-acp/875 connect(fd: 4, uservaddr: 0x7fff8e632870,
                                          addrlen: 110) = -1 ENOENT No such file or directory
2249.435 (0.002 ms): hald-addon-acp/875 close(fd: 4        ) = 0
2249.453 (0.002 ms): hald-addon-acp/875 rt_sigprocmask(how: BLOCK,
                                                       nset: 0x7fff8e6326f0,
                                                       oset: 0x7fff8e632670,
                                                       sigsetsize: 8) = 0
2249.456 (0.002 ms): hald-addon-acp/875 rt_sigaction(sig: CHLD, oact: 0x7fff8e632480,
                                                     sigsetsize: 8) = 0
$
```

Misturando com tracepoints

- Da mesma forma que com as outras ferramentas: –event
- Implementação recente
- A opção -e usada como no strace: nome de syscalls

Mixing with tracepoints

```
$ trace --event sched:sched_process_exec,sched:sched_switch,sched:sched_process_exit sleep 1
$ trace --event sched:sched_process_exec,sched:sched_switch,sched:sched_process_exit sleep 1
    0.050 (         ): sched:sched_process_exec:filename=/usr/bin/sleep pid=3269 old_pid=3269)
    0.079 ( 0.002 ms): sleep/3269 brk(                                  ) = 0x1099000
<SNIP>
    0.515 ( 0.001 ms): sleep/3269 close(fd: 3                           ) = 0
    0.564 ( 0.004 ms): sleep/3269 nanosleep(rqtp: 0x7fffdc44a020        ) ...
    0.564 (         ): sched:sched_switch:sleep:3269 [120] S ==> swapper/5:0 [120])
 1000.762 (1000.201 ms): sleep/3269  ... [continued]: nanosleep()) = 0
 1000.777 ( 0.002 ms): sleep/3269 close(fd: 1                           ) = 0
 1000.781 ( 0.001 ms): sleep/3269 close(fd: 2                           ) = 0
 1000.783 ( 0.000 ms): sleep/3269 exit_group(
 1000.815 (         ): sched:sched_process_exit:comm=sleep pid=3269 prio=120)
$
```

No syscalls

```
$ trace --no-sys --ev sched:*process_exec,sched:*switch,sched:*process_exit sleep 1
    0.048 sched:sched_process_exec:filename=/usr/bin/sleep pid=3279 old_pid=3279)
    0.457 sched:sched_switch:sleep:3279 [120] S ==> swapper/5:0 [120])
 1000.697 sched:sched_process_exit:comm=sleep pid=3279 prio=120)
$
```

Decoração de argumentos de syscalls

- Similar ao strace
- Mas precisa de mais dados só disponíveis no kernel
- Vamos usar o 'perf probe' para obtê-los!

perf probe

- Cria pontos de amostragem dinâmicos
- Em praticamente qualquer lugar
- Devem ser ativados
- Pode coletar valor de variáveis
- Depuração não interativa
- kernel: kprobes
- Userspace: uprobes

Onde colocar um ponto de amostragem

```
# perf probe -L getname_flags
<getname_flags@/usr/src/debug/kernel-3.17.fc20/linux-3.17.8-200.fc20.x86_64/fs/namei.c:0>
      0  getname_flags(const char __user *filename, int flags, int *empty)
      1  {
          struct filename *result, *err;
          int len;
          long max;
          char *kname;
<SNIP>
     25   len = strncpy_from_user(kname, filename, max);
     26   if (unlikely(len < 0)) {
     27   err = ERR_PTR(len);
          goto error;
          }
<SNIP>
     65   result->uptr = filename;
     66   result->aname = NULL;
          audit_getname(result);
          return result;

         error:
     71   final_putname(result);
     72   return err;
     73  }
```

Visualização do código fonte

- Mostra offsets de números de linha na função
- Onde pontos de amostragem podem ser colocados
- Necessita de informação de depuração DWARF
- Pacotes foo-debuginfo ou binários construidos com 'gcc -g'

Inserindo o ponto de amostragem

```
# perf probe 'vfs_getname=getname_flags:65 pathname=filename:string'
Added new event:
  probe:vfs_getname    (on getname_flags:65 with pathname=filename:string)

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

perf record -e probe:vfs_getname -aR sleep 1

# perf probe --list
  probe:vfs_getname    (on getname_flags:65@fs/namei.c with pathname)
```

Tentando utilizá-lo

```
# perf record -e probe:vfs_getname touch My-File-Name
[ perf record: Woken up 1 times to write data ]
# perf script
 touch 880 [2] 0.65794: probe:vfs_getname: (ffff8120b573) pathname="/etc/ld.so.preload"
 touch 880 [2] 0.65802: probe:vfs_getname: (ffff8120b573) pathname="/etc/ld.so.cache"
 touch 880 [2] 0.65817: probe:vfs_getname: (ffff8120b573) pathname="/lib64/libc.so.6"
 touch 880 [2] 0.66003: probe:vfs_getname: (ffff8120b573) pathname="/usr/lib/locale/locale-arc
 touch 880 [2] 0.66048: probe:vfs_getname: (ffff8120b573) pathname="My-File-Name"
#
```

Utilizando-os com o 'trace'

```
# trace --no-sys --ev probe:* touch My-File-Name
 0.149 probe:vfs_getname:(ffff8120b573) pathname="/etc/ld.so.preload")
 0.169 probe:vfs_getname:(ffff8120b573) pathname="/etc/ld.so.cache")
 0.196 probe:vfs_getname:(ffff8120b573) pathname="/lib64/libc.so.6")
 0.552 probe:vfs_getname:(ffff8120b573) pathname="/usr/lib/locale/locale-archive")
 0.651 probe:vfs_getname:(ffff8120b573) pathname="My-File-Name")
#
```

Voltando ao utilitário trace

- o trace usa este ponto de amostragem "vfs_getname", se disponível
- pontenciais tracepoints podem ser prototipados desta forma
- Eventualmente podem se tornar um tracepoint suportado
- Mudanças em kernels podem ser isolados usando uma interface padrão
- Pois a localização no código fonte ou nome da variável podem mudar
- Mas "vfs_getname" e "pathname" isolam estas mudanças

trace using vfs_getname

```
# trace -e open,close,dup2 touch My-File-Name | grep My
 0.991 (0.002 ms): dup2(oldfd: 3<My-File-Name>) = 0
 0.995 (0.001 ms): close(fd: 3<My-File-Name>  ) = 0
#
```

O trace precisa de mais ajuda do probe

- Quando copiando argumentos de syscalls para o kernel
- Como sinalizar quanto copiar?
- Criar automaticamente tais pontos de amostragem?

probe integration with other tools

- Pontos de amostragem são tracepoints
- Todas as outras funcionalidades podem ser usadas: e.g. callchains
- Uso de perl ou python para tratar tais amostras

O ciclo de vida de um pacote de rede ping

```
$ perf probe -L icmp_rcv
<icmp_rcv@/usr/src/debug/kernel-3.17.fc20/linux-3.17.8-200.fc20.x86_64/net/ipv4/icmp.c:0>
      0  int icmp_rcv(struct sk_buff *skb)
      1  {
          struct icmphdr *icmph;
      3   struct rtable *rt = skb_rtable(skb);
          struct net *net = dev_net(rt->dst.dev);
<SNIP>
     28   if (skb_checksum_simple_validate(skb))
          goto csum_error;

     31   if (!pskb_pull(skb, sizeof(*icmph)))
          goto error;

     34   icmph = icmp_hdr(skb);
<SNIP>
     51   if (rt->rt_flags & (RTCF_BROADCAST | RTCF_MULTICAST)) {
          /*
           * RFC 1122: 3.2.2.6 An ICMP_ECHO to broadcast MAY be
           *   silently ignored (we let user decide with a sysctl).
           * RFC 1122: 3.2.2.8 An ICMP_TIMESTAMP MAY be silently
           *   discarded if to broadcast/multicast.
           */
     58   if ((icmph->type == ICMP_ECHO ||
     59        icmph->type == ICMP_TIMESTAMP) &&
```

Inserindo o ponto de amostragem

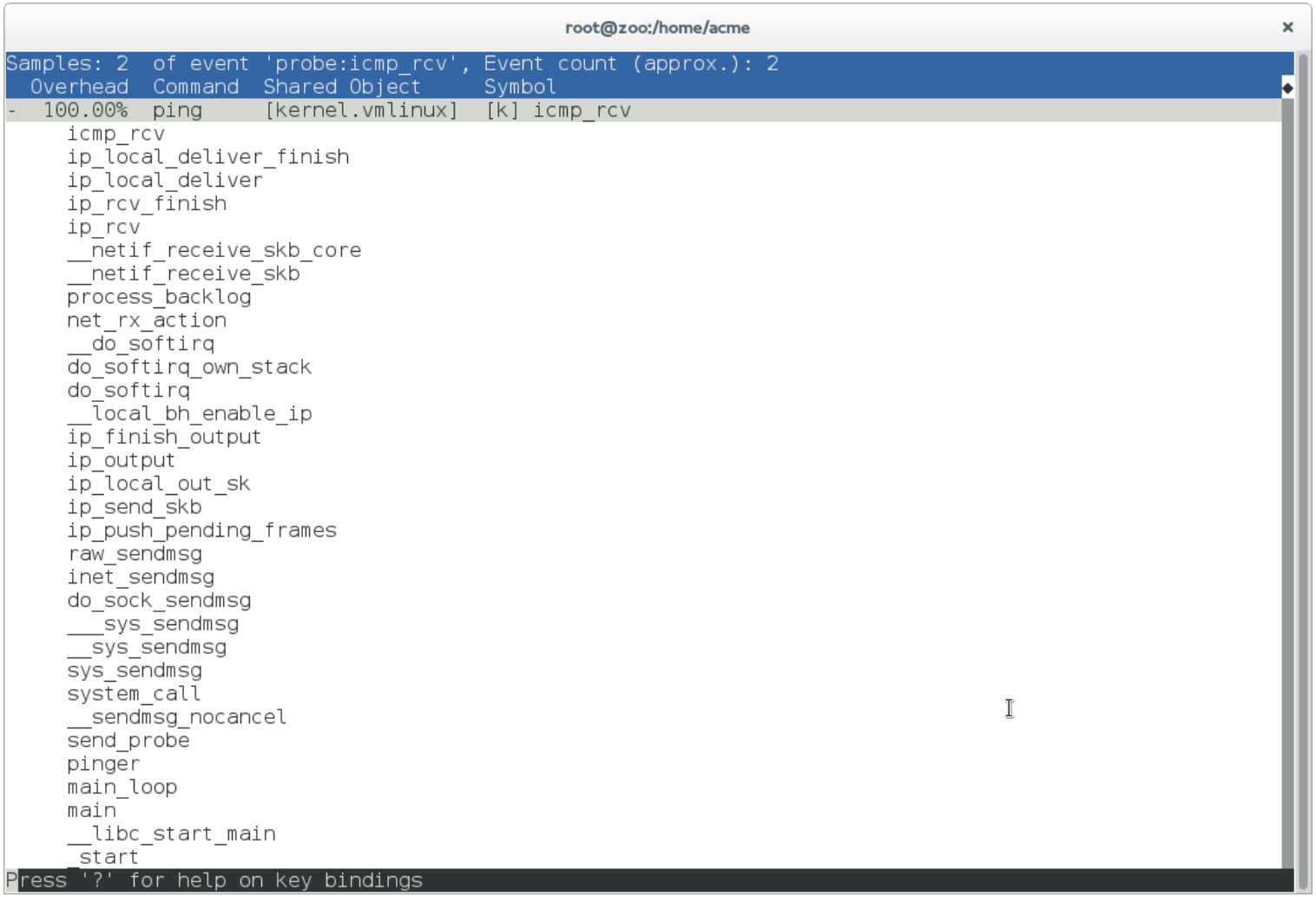
```
# perf probe icmp_rcv:59
Added new event:
  probe:icmp_rcv       (on icmp_rcv:59)

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

perf record -e probe:icmp_rcv -aR sleep 1

#
```

Vamos obter algumas amostras!

```
# perf record -e probe:icmp_rcv --call-graph dwarf ping -b 127.255.255.255
WARNING: pinging broadcast address
PING 127.255.255.255 (127.255.255.255) 56(84) bytes of data.
^C
--- 127.255.255.255 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 999ms

[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.033 MB perf.data (2 samples) ]
#
```

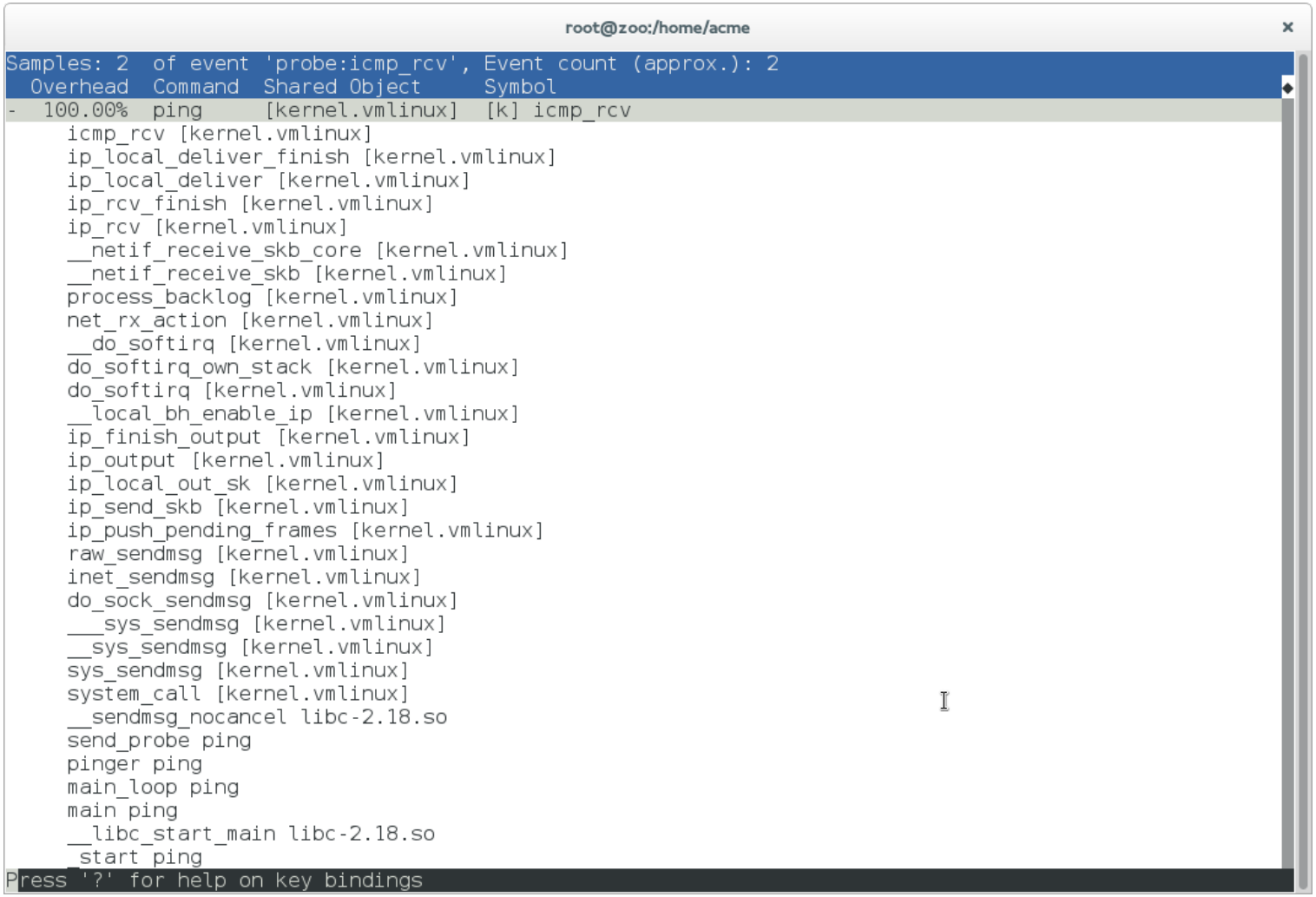
Sequência de eventos do kernel até o programa ping

```
# perf report
```



Mais detalhes - shift+V

```
# perf report
```



Usando scripts para tratar tais amostras

```
# perf script -g python
generated Python script: perf-script.py
# cat perf-script.py
# remove some boilerplate

def probe__icmp_rcv(event_name, context, common_cpu,
                    common_secs, common_nsecs, common_pid, common_comm,
                    common_callchain, __probe_ip):

    print "__probe_ip=%u" % (__probe_ip)

    for node in common_callchain:
        if 'sym' in node:
            print "\t[%x] %s" % (node['ip'], node['sym']['name'])
        else:
            print "        [%x]" % (node['ip'])

    print "\n"
#
```

Listando funções amostráveis em uma biblioteca

```
# perf probe -F /lib64/libc-2.12.so|grep ^m|head -10
madvise
malloc
malloc@plt
malloc_info
mblen
mbstowcs
mbtowc
mcheck
mcheck_check_all
mcheck_pedantic
#
```

Adicionando um ponto de amostragem

```
# perf probe -x /lib64/libc-2.12.so malloc
Added new event:
  probe_libc:malloc    (on 0x79b80)

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

perf record -e probe_libc:malloc -aR sleep 1

#
```

Coletando callchains com pedaços da pilha

```
# perf record -e probe_libc:* -g dwarf,1024 sleep 2
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.058 MB perf.data (~2547
#
```

snapshot do report

```
# cat perf.hist.5
-  100.00%  sleep  libc-2.12.so  [.] malloc
   - malloc
      - 45.16% __strdup
         + 85.71% setlocale
         + 7.14% _nl_load_locale_from_archive
         + 7.14% __textdomain
      + 38.71% _nl_intern_locale_data
      + 6.45% _nl_normalize_codeset
      + 3.23% _nl_load_locale_from_archive
      - 3.23% new_composite_name
           setlocale
           0x4014ec
           __libc_start_main
           0x4011f9
      + 3.23% set_binding_values
#
```

snapshot com mais detalhes

```
# cat perf.hist.6
-  100.00%  sleep  libc-2.12.so  [.] malloc
   - malloc libc-2.12.so
      - 45.16% __strdup libc-2.12.so
         + 85.71% setlocale libc-2.12.so
         + 7.14% _nl_load_locale_from_archive libc-2.12.so
         + 7.14% __textdomain libc-2.12.so
      + 38.71% _nl_intern_locale_data libc-2.12.so
      + 6.45% _nl_normalize_codeset libc-2.12.so
      + 3.23% _nl_load_locale_from_archive libc-2.12.so
      - 3.23% new_composite_name libc-2.12.so
           setlocale libc-2.12.so
           0x4014ec sleep
           __libc_start_main libc-2.12.so
           0x4011f9 sleep
      + 3.23% set_binding_values libc-2.12.so
# rpm -qf `which sleep`
coreutils-8.4-19.el6.x86_64
# rpm -q coreutils-debuginfo
package coreutils-debuginfo is not installed
# rpm -q glibc-debuginfo
glibc-debuginfo-2.12-1.80.el6_3.4.x86_64
#
```

Obtendo traces com tais pontos de amostra

```
# trace --ev sched:*exec,probe*:* usleep 1
  0.051 (         ): sched:sched_process_exec:filename=/bin/usleep pid=30315 old_pid=30315
  0.072 (0.002 ms): brk(                                                    ) = 0x1a4e000
  0.085 (0.003 ms): mmap(len: 4096, prot: RD|WR, flags: PRIV|ANON, fd: -1) = 0x7fc4b308b000
  0.105 (0.002 ms): open(filename: 0x7fc4b2e88048, flags: CLOEXEC           ) ...
  0.105 (         ): probe:vfs_getname:(ffff811ed023) pathname="/etc/ld.so.cache")
  0.109 (0.006 ms):  ... [continued]: open()) = 3
  0.112 (0.002 ms): fstat(fd: 3</etc/ld.so.cache>, statbuf: 0x7fffbaa9d120) = 0
  0.119 (0.001 ms): close(fd: 3</etc/ld.so.cache>                           ) = 0
  0.128 (0.002 ms): open(filename: 0x7fc4b30806a1, flags: CLOEXEC           ) ...
  0.128 (         ): probe:vfs_getname:(ffff811ed023) pathname="/lib64/libpopt.so.0")
<SNIP>
  0.314 (0.003 ms): mprotect(start: 0x7fc4b308c000, len: 4096, prot: READ ) = 0
  0.323 (0.007 ms): munmap(addr: 0x7fc4b3070000, len: 107451               ) = 0
  0.341 (         ): probe_libc:malloc:(7fc4b2922ca0))
  0.376 (         ): probe_libc:malloc:(7fc4b2922ca0))
  0.384 (0.001 ms): brk(                                                    ) = 0x1a4e000
  0.388 (0.002 ms): brk(brk: 0x1a6f000                                      ) = 0x1a6f000
  0.390 (0.001 ms): brk(                                                    ) = 0x1a6f000
  0.395 (         ): probe_libc:malloc:(7fc4b2922ca0))
  0.405 (         ): probe_libc:malloc:(7fc4b2922ca0))
  0.479 (0.066 ms): nanosleep(rqtp: 0x7fffbaa9d8f0                          ) = 0
  0.489 (0.000 ms): exit_group(
```

That is all folks!

Muito obrigado!

Arnaldo Carvalho de Melo

acme@kernel.org

acme@redhat.com

http://vger.kernel.org/ acme/perf/cinfotec2015.pdf

http://perf.wiki.kernel.org/

linux-perf-users@vger.kernel.org