Introduction to Linux Tracing and its Concepts

Elena Zannoni
Elena.zannoni@oracle.com
ezannoni@gmail.com

In the Beginning...

- Ptrace() system call
- Used by debuggers to control the process being debugged
- Used by strace()
- Can do many actions
  - Start process
  - Attach to process
  - Execute process
  - Read / write memory
  - Read / write registers

strace

```
[root@fedora ~]# strace -e openat,write echo HELLO
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
write(1, "HELLO\n", 6HELLO
)                  = 6
+++ exited with 0 +++
[root@fedora ~]# strace -c echo HELLO
HELLO
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 53.62    0.000348         348         1           execve
 11.25    0.000073           8         9           mmap
  6.01    0.000039           9         4           mprotect
  4.93    0.000032          10         3           openat
  4.31    0.000028           5         5           close
  4.01    0.000026          26         1           write
  4.01    0.000026           6         4           newfstatat
  3.39    0.000022           5         4           pread64
  2.47    0.000016           5         3           brk
  2.00    0.000013          13         1           munmap
  1.54    0.000010           5         2         1 arch_prctl
  1.39    0.000009           9         1         1 access
  1.08    0.000007           7         1           read
------ ----------- ----------- --------- --------- ----------------
100.00    0.000649          16        39         2 total
```

Classical Debugging Session

- Run program to a specific point in the code
- When stopped: print information, usually variable values, backtraces
- Sometimes also set information (to test a potential fix for instance,”what if X was 2”)
- Interactive, under controlled environment

What is a breakpoint?

- A way to stop program execution at a certain instruction
- Used by debuggers
- Substitute original program instruction with illegal instruction (or specific BP instruction, depending on architecture)
- Reaching it during program execution will generate an exception
- At that point the debugger takes control, allowing to interactively inspect the program state.
- Ptrace !!

Profiling

- Statistical, sampling at a certain frequency
- Typically interested in PMU events
- Events to record are fixed

Tracing

- Run unperturbed, minimal overhead
- Collect information at certain points in the program
- “Manipulate” information before presenting to user
- Display the information collected
- Similar but not the same as debugging
- Can dynamically trace points of interest

~~Brief~~ History

- Almost 20 years of Linux tracing! We are all getting older.
- Tracing in Linux was non existant until the mid 2000’s.
- Took a while to be acknowledged as a real user need
- Developers worried about overhead, slowdown…
- Developers feared of being locked into an ABI
- Eventually pieces started being added, fragmented approach
- LTT (Linux Trace Toolkit) (1998)
- Kprobes (2004): http://www.ibm.com/developerworks/library/l-kprobes/index.html
- Systemtap for Linux (2005) at OLS https://www.kernel.org/doc/ols/2005/ols2005v2-pages-57-72.pdf
- LTTng (LTT Next Generation) (2006): https://lore.kernel.org/lkml/20060109175234.GB19850@Krystal/
- Ftrace: (2008): https://lore.kernel.org/lkml/20080103071609.478486470@goodmis.org/
- Perf: (2008) https://lore.kernel.org/lkml/20081204225345.654705757@linutronix.de/
- DTrace for Linux (2011)
- (e)BPF: 2013

What Infrastructure

- Need to be able to specify points of interest in execution of program
- Need to be able to specify what information is needed at those points
- Need to process information collected
- Need to pass the result to user somehow

Probes

- Goal: associate actions to be performed at specific addresses reached by program execution
- Action is generally : collect information, process information
- Types of probes
  - Kprobes
  - Kretprobes
  - Uprobes
  - Uretprobes

kprobes

- Used for tracing of running kernel
- Kernel must be configured with CONFIG_KPROBES=y
- Main concept is similar to debugger breakpoints: place breakpoint instruction at desired code location
- When hit, exception is caused
- Exception handler executes actions associated with kprobe
- Optimizations to kprobes using Jumps instead of exceptions
- Used by all tracing tools

uprobes

- Implementation based on inodes
- Must be enabled with CONFIG_UPROBES
- Uprobes described as: inode (file), offset in file (map), list of associated actions, arch specific info (for instruction handling)
- Probes stored in an rb_tree
- Register a uprobe: add probe to probe tree (if needed), and insert the arch specific BP instruction
- Handle the uprobe by calling the actions
- Resume to userspace
- Multiple consumers per probe allowed (ref count used)
- Conditional execution of actions is possible (filtering)

Kretprobes & Uretprobes

- Place probes at exit of functions
- Done in two steps:
- Place probe at entry
- When this is hit, its handler places retprobe at return address
- Note: retprobe location is after called function ends

Tracepoints

- Aka Statically defined tracing (SDT)
- Static probe points in kernel code
- Added by kernel subsystem maintainers. Many exist in the kernel in various subsystems, and being added.
- Syntax is independent of users (many tools read them and use them)
- Definitions in the kernel file: include/linux/tracepoint.h
- Need 2 pieces
- Define actions to be executed.
- Two ways (see include/trace/events/*.h) :
- TRACE_EVENT(...) for a single event
- DEFINE_EVENT(...) and DECLARE_EVENT_CLASS(...) for multiple events with similar structure
- Mark tracing locations with function calls like trace_<my_event_name>(…)

include/trace/events/alarmtimer.h

```
DEFINE_EVENT(alarm_class, alarmtimer_fired,

  TP_PROTO(struct alarm *alarm, ktime_t now),

  TP_ARGS(alarm, now)

);


DEFINE_EVENT(alarm_class, alarmtimer_start,

  TP_PROTO(struct alarm *alarm, ktime_t now),

  TP_ARGS(alarm, now)

);
```

```
DECLARE_EVENT_CLASS(alarm_class,
   TP_PROTO(struct alarm *alarm, ktime_t now),
   TP_ARGS(alarm, now),
   TP_STRUCT__entry(
            __field(void *, alarm)
            __field(unsigned char, alarm_type)
            __field(s64, expires)
            __field(s64, now)
   ),
   TP_fast_assign(
            __entry->alarm = alarm;
            __entry->alarm_type = alarm->type;
            __entry->expires = alarm->node.expires;
            __entry->now = now;
    ),
    TP_printk("alarmtimer:%p type:%s expires:%llu now:%llu",
              __entry->alarm,
              show_alarm_type((1 << __entry->alarm_type)),
              __entry->expires,
              __entry->now
    )
);
```

kernel/time/alarmtimer.c

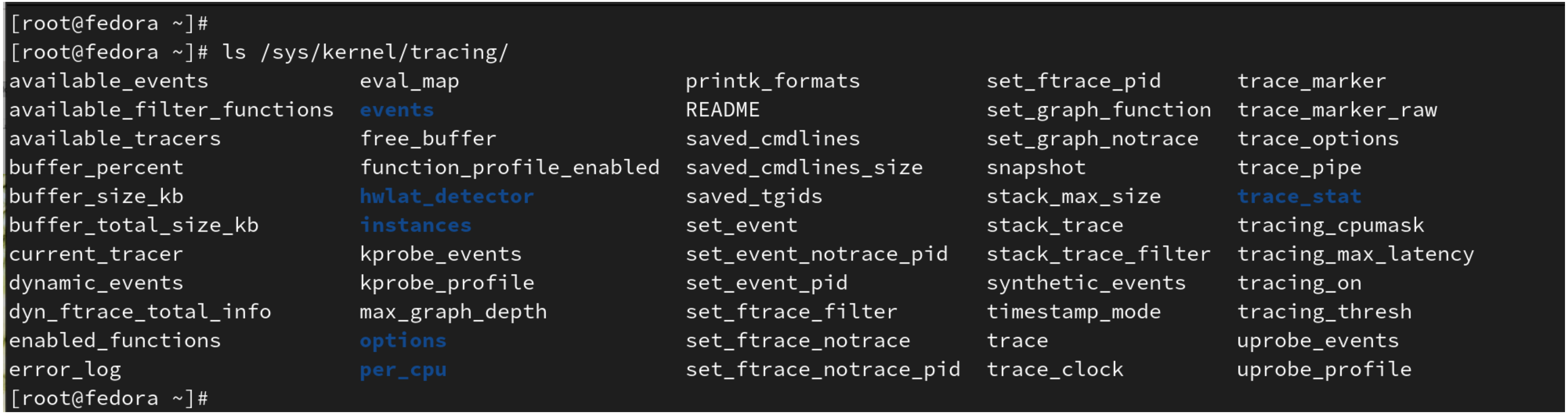
```
/**
 * alarmtimer_fired - Handles alarm hrtimer being fired.
 * @timer: pointer to hrtimer being run
 * When an alarm timer fires, this runs through the timerqueue to
 * see which alarms expired, and runs those. If there are more alarm
 * timers queued for the future, we set the hrtimer to fire when
 * the next future alarm timer expires.
 */
static enum hrtimer_restart alarmtimer_fired(struct hrtimer *timer)
{
  struct alarm *alarm = container_of(timer, struct alarm, timer);
  struct alarm_base *base = &alarm_bases[alarm->type];

  [...do stuff...]

trace_alarmtimer_fired(alarm, base->get_ktime());
  return ret;
}
```

```
/**
 * alarm_start - Sets an absolute alarm to fire
 * @alarm: ptr to alarm to set
 * @start: time to run the alarm
 */
void alarm_start(struct alarm *alarm, ktime_t start)
{
    struct alarm_base *base = &alarm_bases[alarm->type];
    unsigned long flags;

    spin_lock_irqsave(&base->lock, flags);
    alarm->node.expires = start;
    alarmtimer_enqueue(base, alarm);
    hrtimer_start(&alarm->timer, alarm->node.expires, HRTIMER_MODE_ABS);
    spin_unlock_irqrestore(&base->lock, flags);

    trace_alarmtimer_start(alarm, base->get_ktime());
}
```

TraceFS (1)

- Tracefs pseudo filesystem: /sys/kernel/tracing
- Mounted if kernel FTRACE config options are set, like CONFIG_FTRACE=y (check in /boot/config-<kernel-version> on Fedora)
- Many files to control ftrace behavior, what to trace, turn tracing on/off

TraceFS (2)



```
[root@fedora ~]#
[root@fedora ~]# ls /sys/kernel/tracing/
available_events            eval_map                  printk_formats          set_ftrace_pid      trace_marker
available_filter_functions  events                    README                  set_graph_function  trace_marker_raw
available_tracers           free_buffer               saved_cmdlines          set_graph_notrace   trace_options
buffer_percent              function_profile_enabled  saved_cmdlines_size     snapshot            trace_pipe
buffer_size_kb              hwlat_detector            saved_tgids             stack_max_size      trace_stat
buffer_total_size_kb        instances                 set_event               stack_trace         tracing_cpumask
current_tracer              kprobe_events             set_event_notrace_pid   stack_trace_filter  tracing_max_latency
dynamic_events              kprobe_profile            set_event_pid           synthetic_events    tracing_on
dyn_ftrace_total_info       max_graph_depth           set_ftrace_filter       timestamp_mode      tracing_thresh
enabled_functions           options                   set_ftrace_notrace      trace               uprobe_events
error_log                   per_cpu                   set_ftrace_notrace_pid  trace_clock         uprobe_profile
[root@fedora ~]#
```

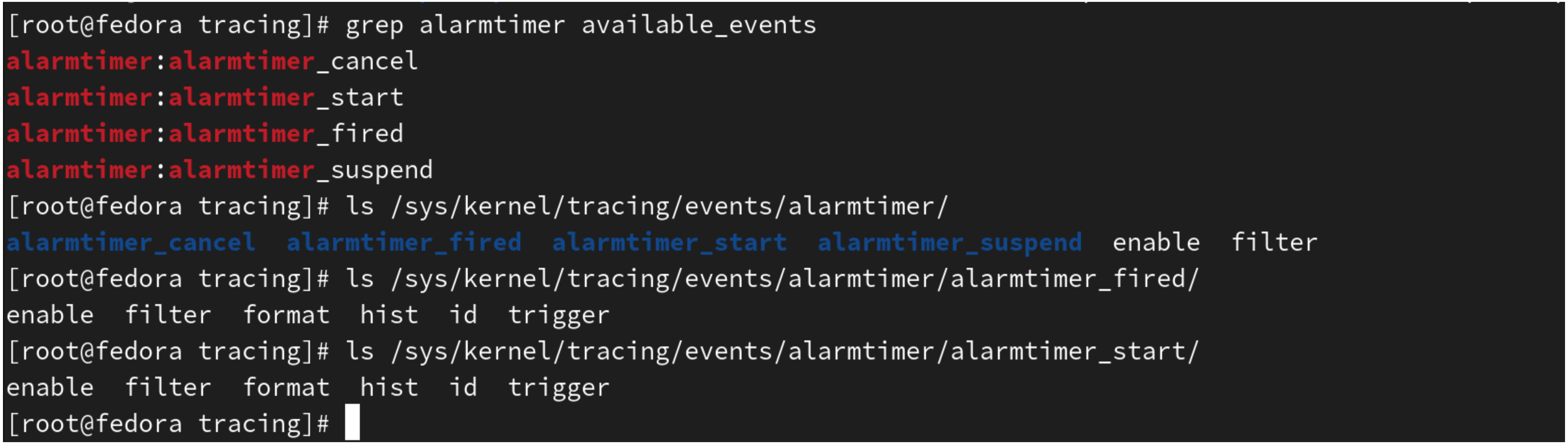
What do I do with all this?

- Many tools on top of this infrastructure
- Static vs dynamic tracing

FTrace

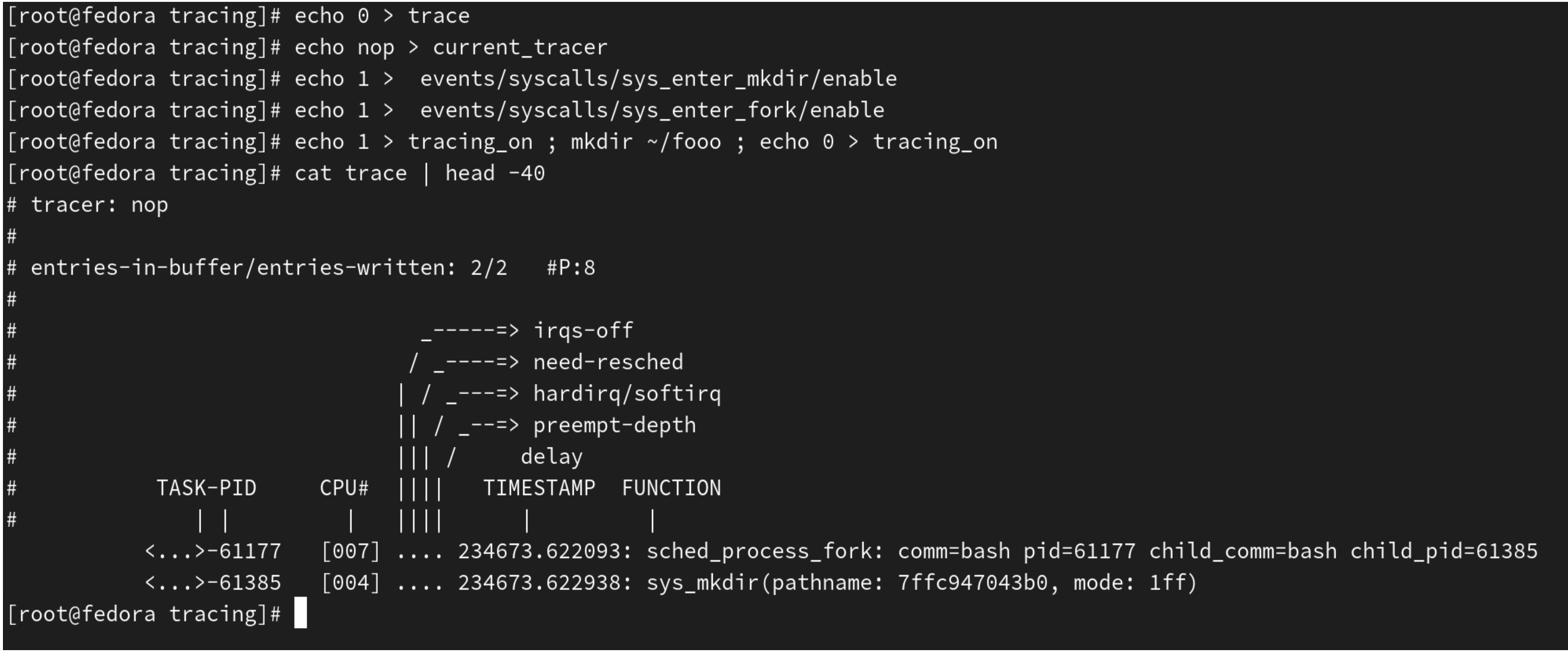
- Kernel tracer. Monitor many different areas and activities in the kernel
- Interface: via /sys/kernel/debug/tracing (both control and output)
- Documentation in kernel tree: Documentation/trace/ftrace.txt and ftrace_design.txt
- current_tracer: which tracer is in effect (could be NOP)
- tracing_on: writing to buffer is enabled
- trace: the output buffer (circular, will overwrite)
- trace_pipe: output from live tracing
- available_events: which events (static points in kernel) are available
- available_tracers: which tracers are available (relates to kconfig options, for instance function_graph, function, nop...)
- kprobe_events, uprobe_events: written to when a kprobe (uprobe) is placed, empty if none
- options, instances, events, per_cpu, trace_stats: directories
- [....]

Tracefs: static events



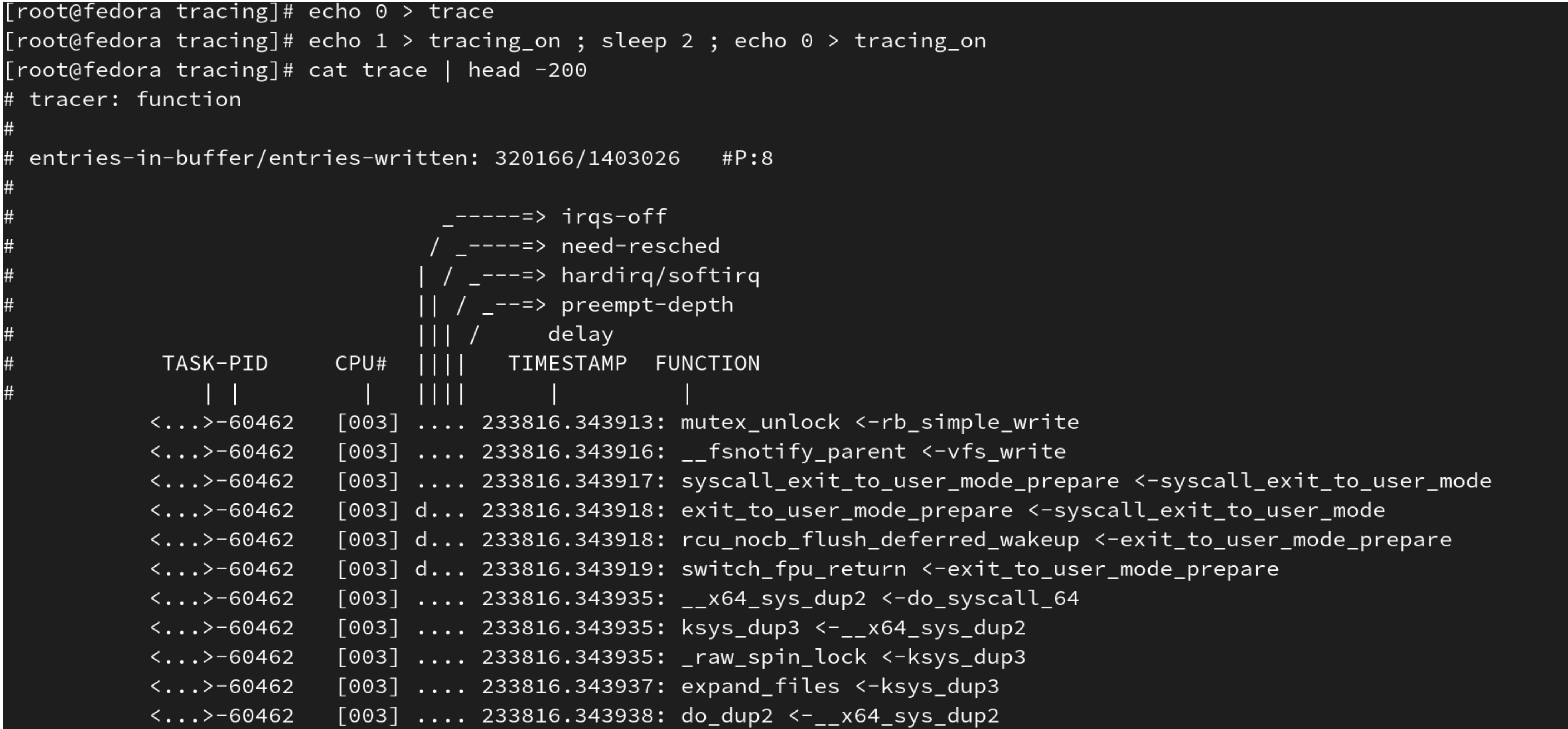
```
[root@fedora tracing]# grep alarmtimer available_events
alarmtimer:alarmtimer_cancel
alarmtimer:alarmtimer_start
alarmtimer:alarmtimer_fired
alarmtimer:alarmtimer_suspend
[root@fedora tracing]# ls /sys/kernel/tracing/events/alarmtimer/
alarmtimer_cancel  alarmtimer_fired  alarmtimer_start  alarmtimer_suspend  enable  filter
[root@fedora tracing]# ls /sys/kernel/tracing/events/alarmtimer/alarmtimer_fired/
enable  filter  format  hist  id  trigger
[root@fedora tracing]# ls /sys/kernel/tracing/events/alarmtimer/alarmtimer_start/
enable  filter  format  hist  id  trigger
[root@fedora tracing]# 
```

A Simple Example



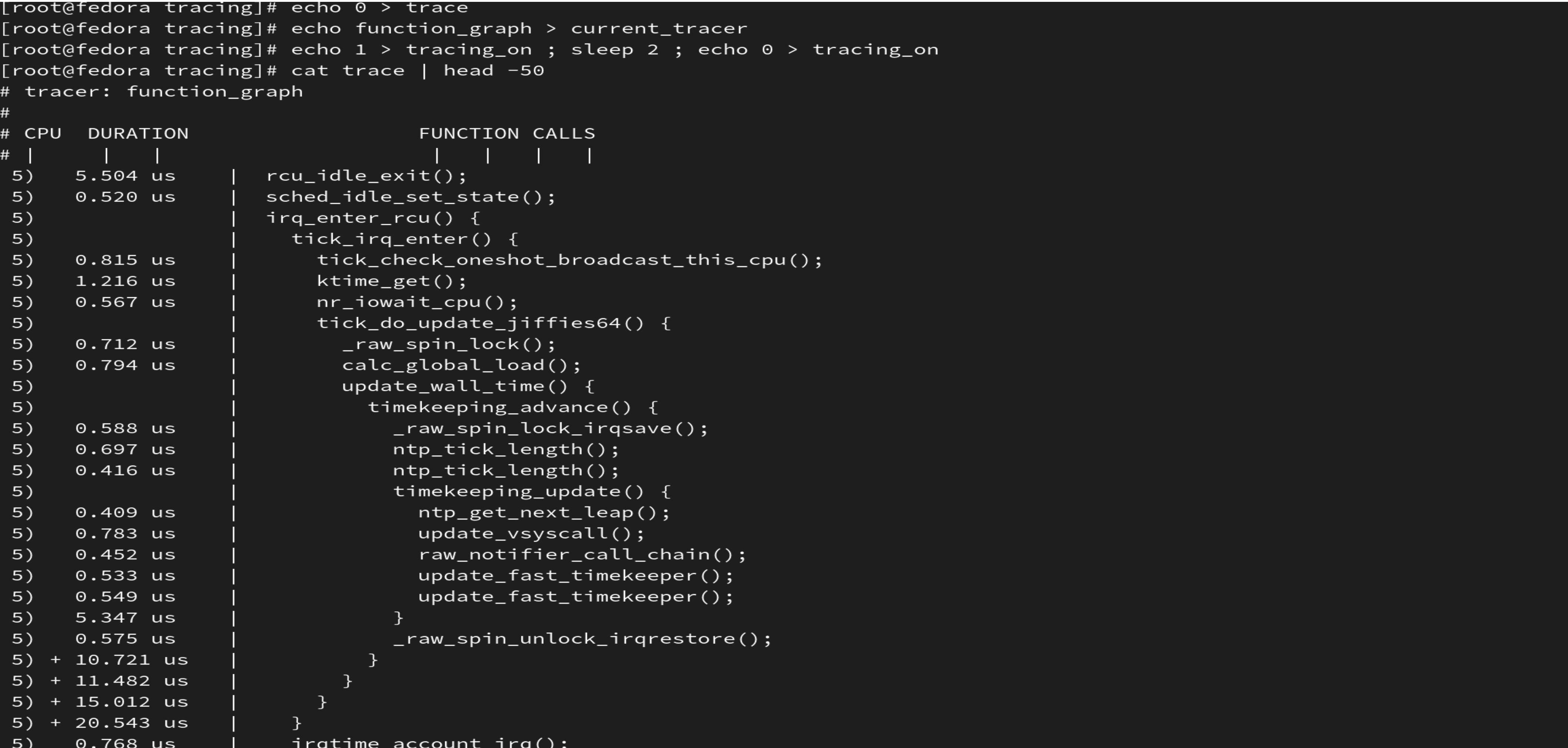
```
[root@fedora tracing]# echo 0 > trace
[root@fedora tracing]# echo nop > current_tracer
[root@fedora tracing]# echo 1 >  events/syscalls/sys_enter_mkdir/enable
[root@fedora tracing]# echo 1 >  events/syscalls/sys_enter_fork/enable
[root@fedora tracing]# echo 1 > tracing_on ; mkdir ~/fooo ; echo 0 > tracing_on
[root@fedora tracing]# cat trace | head -40
# tracer: nop
#
# entries-in-buffer/entries-written: 2/2   #P:8
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
           <...>-61177   [007] .... 234673.622093: sched_process_fork: comm=bash pid=61177 child_comm=bash child_pid=61385
           <...>-61385   [004] .... 234673.622938: sys_mkdir(pathname: 7ffc947043b0, mode: 1ff)
[root@fedora tracing]# 
```

Function Tracer



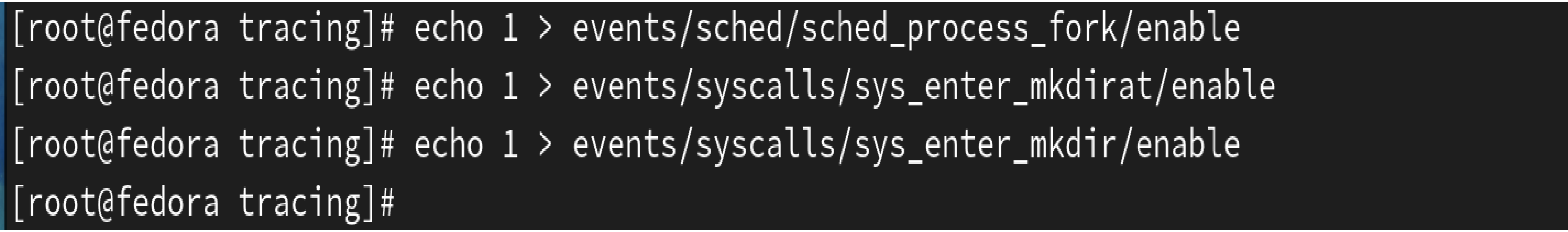
```
[root@fedora tracing]# echo 0 > trace
[root@fedora tracing]# echo 1 > tracing_on ; sleep 2 ; echo 0 > tracing_on
[root@fedora tracing]# cat trace | head -200
# tracer: function
#
# entries-in-buffer/entries-written: 320166/1403026   #P:8
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
           <...>-60462   [003] .... 233816.343913: mutex_unlock <-rb_simple_write
           <...>-60462   [003] .... 233816.343916: __fsnotify_parent <-vfs_write
           <...>-60462   [003] .... 233816.343917: syscall_exit_to_user_mode_prepare <-syscall_exit_to_user_mode
           <...>-60462   [003] d... 233816.343918: exit_to_user_mode_prepare <-syscall_exit_to_user_mode
           <...>-60462   [003] d... 233816.343918: rcu_nocb_flush_deferred_wakeup <-exit_to_user_mode_prepare
           <...>-60462   [003] d... 233816.343919: switch_fpu_return <-exit_to_user_mode_prepare
           <...>-60462   [003] .... 233816.343935: __x64_sys_dup2 <-do_syscall_64
           <...>-60462   [003] .... 233816.343935: ksys_dup3 <-__x64_sys_dup2
           <...>-60462   [003] .... 233816.343935: _raw_spin_lock <-ksys_dup3
           <...>-60462   [003] .... 233816.343937: expand_files <-ksys_dup3
           <...>-60462   [003] .... 233816.343938: do_dup2 <-__x64_sys_dup2
```

Function Graph Tracer



```
[root@fedora tracing]# echo 0 > trace
[root@fedora tracing]# echo function_graph > current_tracer
[root@fedora tracing]# echo 1 > tracing_on ; sleep 2 ; echo 0 > tracing_on
[root@fedora tracing]# cat trace | head -50
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 5)   5.504 us    |  rcu_idle_exit();
 5)   0.520 us    |  sched_idle_set_state();
 5)               |  irq_enter_rcu() {
 5)               |    tick_irq_enter() {
 5)   0.815 us    |      tick_check_oneshot_broadcast_this_cpu();
 5)   1.216 us    |      ktime_get();
 5)   0.567 us    |      nr_iowait_cpu();
 5)               |      tick_do_update_jiffies64() {
 5)   0.712 us    |        _raw_spin_lock();
 5)   0.794 us    |        calc_global_load();
 5)               |        update_wall_time() {
 5)               |          timekeeping_advance() {
 5)   0.588 us    |            _raw_spin_lock_irqsave();
 5)   0.697 us    |            ntp_tick_length();
 5)   0.416 us    |            ntp_tick_length();
 5)               |            timekeeping_update() {
 5)   0.409 us    |              ntp_get_next_leap();
 5)   0.783 us    |              update_vsyscall();
 5)   0.452 us    |              raw_notifier_call_chain();
 5)   0.533 us    |              update_fast_timekeeper();
 5)   0.549 us    |              update_fast_timekeeper();
 5)   5.347 us    |            }
 5)   0.575 us    |            _raw_spin_unlock_irqrestore();
 5) + 10.721 us   |          }
 5) + 11.482 us   |        }
 5) + 15.012 us   |      }
 5) + 20.543 us   |    }
 5)   0.768 us    |    irqtime_account_irq();
```

Enable a few Static Tracepoints

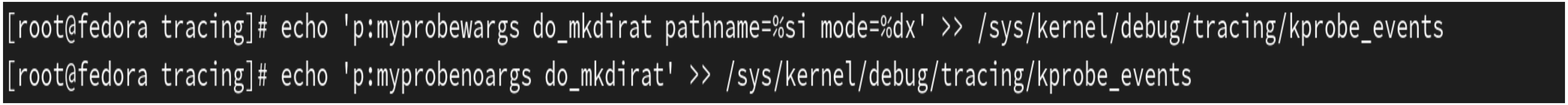


```
[root@fedora tracing]# echo 1 > events/sched/sched_process_fork/enable
[root@fedora tracing]# echo 1 > events/syscalls/sys_enter_mkdirat/enable
[root@fedora tracing]# echo 1 > events/syscalls/sys_enter_mkdir/enable
[root@fedora tracing]#
```

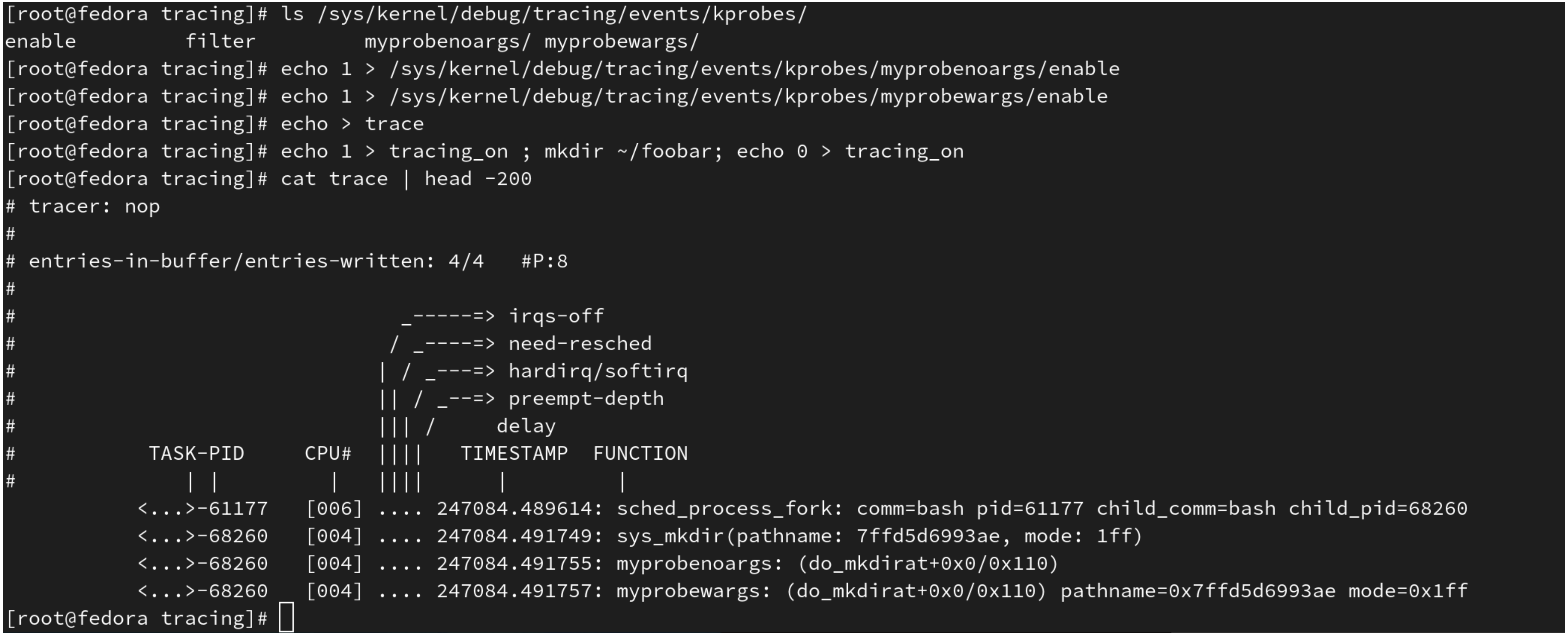
TraceFS and kprobes/uprobes

- Use /sys/kernel/debug/tracing/kprobe_events and /sys/kernel/debug/tracing/uprobe_events to control from command line
- Read more: Documentation/trace/kprobetrace.txt and uprobetracer.txt
- LWN article: http://lwn.net/Articles/343766/
-
- Set kretprobe:
- **echo 'r:myretprobe do_sys_open $retval' > /sys/kernel/debug/tracing/kprobe_events**
-
- Set uprobe:
- **echo 'p: /bin/bash:0x4245c0' > /sys/kernel/debug/tracing/uprobe_events**
-
- Clear them:
- **echo > /sys/kernel/debug/tracing/kprobe_events**
- **echo > /sys/kernel/debug/tracing/uprobe_events**

Set kprobes



```
[root@fedora tracing]# echo 'p:myprobewargs do_mkdirat pathname=%si mode=%dx' >> /sys/kernel/debug/tracing/kprobe_events
[root@fedora tracing]# echo 'p:myprobenoargs do_mkdirat' >> /sys/kernel/debug/tracing/kprobe_events
```



```
[root@fedora tracing]# ls /sys/kernel/debug/tracing/events/kprobes/
enable         filter         myprobenoargs/ myprobewargs/
[root@fedora tracing]# echo 1 > /sys/kernel/debug/tracing/events/kprobes/myprobenoargs/enable
[root@fedora tracing]# echo 1 > /sys/kernel/debug/tracing/events/kprobes/myprobewargs/enable
[root@fedora tracing]# echo > trace
[root@fedora tracing]# echo 1 > tracing_on ; mkdir ~/foobar; echo 0 > tracing_on
[root@fedora tracing]# cat trace | head -200
# tracer: nop
#
# entries-in-buffer/entries-written: 4/4   #P:8
#
#                                _-----=> irqs-off
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| /     delay
#           TASK-PID     CPU#  ||||   TIMESTAMP  FUNCTION
#              | |         |   ||||      |         |
           <...>-61177   [006] .... 247084.489614: sched_process_fork: comm=bash pid=61177 child_comm=bash child_pid=68260
           <...>-68260   [004] .... 247084.491749: sys_mkdir(pathname: 7ffd5d6993ae, mode: 1ff)
           <...>-68260   [004] .... 247084.491755: myprobenoargs: (do_mkdirat+0x0/0x110)
           <...>-68260   [004] .... 247084.491757: myprobewargs: (do_mkdirat+0x0/0x110) pathname=0x7ffd5d6993ae mode=0x1ff
[root@fedora tracing]# 
```

perf

- In kernel (tools/perf directory) userspace tool
- Started in 2008 as hardware performance counters interface, initially called perf counters.
- Has grown into all encompassing tracing system. Still very active
- Interfaces to display output: command line, TUI, GUI
- Documentation: tools/perf/Documentation
- Note: Install kernel debugging information RPM (“yum –enablerepo=updates-debuginfo install kernel-debuginfo” on Fedora)

Perf Subcommands

- *Perf stat:* collects and display events data (performance counters) during a command execution
- *Perf record*: run a command, store its profiling (sampling mode) in output file (perf.data) (no output is produced)
- *Perf report*: display data previously recorded in output file (perf.data)
- *Perf diff*: diff between perf.data files
- *Perf top*: performance counter profile in real time (live)
- *Perf probe*: define dynamic tracepoints
- [more...]

List Functions to Probe

```
[root@fedora ~]# perf probe -F *writepages*
[...long output...]
blkdev_writepages
btree_writepages
btrfs_writepages
do_writepages
ext4_dax_writepages
ext4_writepages
extent_writepages
generic_writepages
iomap_writepages
mpage_writepages
[...more...]
```

List Source Code of Function

```
[root@fedora ~]# perf probe -L do_writepages
<do_writepages@/usr/src/debug/kernel-5.11.16/linux-5.11.16-300.fc34.x86_64/mm/page-writeback.c:0>
      0  int do_writepages(struct address_space *mapping, struct writeback_control *wbc)
         {
      2         int ret;

                if (wbc->nr_to_write <= 0)
                        return 0;
      6         while (1) {
                        if (mapping->a_ops->writepages)
      8                         ret = mapping->a_ops->writepages(mapping, wbc);
                        else
     10                         ret = generic_writepages(mapping, wbc);
     11                 if ((ret != -ENOMEM) || (wbc->sync_mode != WB_SYNC_ALL))
                                break;
     13                 cond_resched();
     14                 congestion_wait(BLK_RW_ASYNC, HZ/50);
                }
                return ret;
         }
```

Attempt to Set a Probe

```
[root@fedora ~]# perf probe -V do_writepages
Available variables at do_writepages
        @<do_writepages+0>
                struct address_space*   mapping
                struct writeback_control*       wbc

[root@fedora ~]# perf probe do_writepages:7 wbc
This line is sharing the address with other lines.
Please try to probe at do_writepages:6 instead.
  Error: Failed to add events.


[root@fedora ~]# perf probe 'do_writepages wbc'
Added new events:
  probe:do_writepages (on do_writepages with wbc)

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

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

Did I Really Set a Probe?

```
[root@fedora ~]# perf probe -l
  kprobes:myprobenoargs (on do_mkdirat@fs/namei.c)
  kprobes:myprobewargs (on do_mkdirat@fs/namei.c with pathname mode)
  probe:do_mkdirat     (on do_mkdirat@fs/namei.c with dfd pathname mode path)
  probe:do_writepages  (on do_writepages@mm/page-writeback.c with wbc)
```

```
[root@fedora ~]# ls /sys/kernel/debug/tracing/events/kprobes
enable  filter  myprobenoargs  myprobewargs
```

```
[root@fedora ~]# ls -l /sys/kernel/debug/tracing/events/probe/
total 0
drwxr-xr-x. 2 root root 0 May  2 16:13 do_mkdirat
drwxr-xr-x. 2 root root 0 May  2 17:27 do_writepages
-rw-r--r--. 1 root root 0 May  2 16:08 enable
-rw-r--r--. 1 root root 0 May  2 16:08 filter
```

Are we Sure?

```
[root@fedora ~]# perf list | grep do_writepages
  probe:do_writepages                                [Tracepoint event]
```

```
[root@fedora ~]# perf list | grep probe
  cfg80211:cfg80211_probe_status                     [Tracepoint event]
  cfg80211:rdev_probe_client                         [Tracepoint event]
  cfg80211:rdev_probe_mesh_link                      [Tracepoint event]
  kprobes:myprobenoargs                              [Tracepoint event]
  kprobes:myprobewargs                               [Tracepoint event]
  probe:do_mkdirat                                   [Tracepoint event]
  probe:do_writepages                                [Tracepoint event]
  tcp:tcp_probe                                      [Tracepoint event]
```

```
[root@fedora ~]# cat /sys/kernel/debug/tracing/kprobe_events
p:kprobes/myprobewargs do_mkdirat pathname=%si mode=%dx
p:kprobes/myprobenoargs do_mkdirat
p:probe/do_mkdirat _text+3404112 dfd=%di:s32 pathname=%si:x64 mode=%dx:x16 path=-64(%sp):x64
p:probe/do_writepages _text+2565328 wbc=%si:x64
```

Let’s Try

```
[root@fedora ~]# cat commands.sh
#!/bin/sh
echo 1 > foo.txt
sync
mkdir ./foooooo

[root@fedora ~]# perf record -e probe:* -aRg /bin/sh ./commands.sh
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.333 MB perf.data (10 samples) ]
```

Did it Trigger? (part 1)

```
[root@fedora ~]# perf script
sh 92906 [002] 319527.610159: probe:do_writepages: (ffffffffa42724d0) wbc=0xffffb93c01893e40
        ffffffffa42724d1 do_writepages+0x1 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4269187 __filemap_fdatawrite_range+0xa7 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
        ffffffffa43fb74f ext4_release_file+0x4f
(/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa432eef4 __fput+0x94 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa40fb1d5 task_work_run+0x65 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4164b91 exit_to_user_mode_prepare+0x181 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
        ffffffffa4bcb918 syscall_exit_to_user_mode+0x18 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
        ffffffffa4c0008c entry_SYSCALL_64_after_hwframe+0x44 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
            7fc4753c0f3b __dup2+0xb (/usr/lib64/libc-2.33.so)
            55b90f52269e do_redirections+0x9e (/usr/bin/bash)
                       1 [unknown] ([unknown])
                       a [unknown] ([unknown])



[...more output...]
```

Did it Trigger? (part 2)

```
[...continued...]


sync 92908 [005] 319527.617497: probe:do_writepages: (ffffffffa42724d0) wbc=0xffffb93c0631be80
        ffffffffa42724d1 do_writepages+0x1 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4269187 __filemap_fdatawrite_range+0xa7 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
        ffffffffa437645f iterate_bdevs+0xaf (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa436860e ksys_sync+0x5e (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa436864a __ia32_sys_sync+0xa (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4bc7a33 do_syscall_64+0x33 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4c0008c entry_SYSCALL_64_after_hwframe+0x44 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
            7f724f3563cb sync+0xb (/usr/lib64/libc-2.33.so)

mkdir 92909 [000] 319527.618401:    probe:do_mkdirat: (ffffffffa433f150) dfd=-100 pathname=0x7ffd683c4385
mode=0x1ff path=0x0
        ffffffffa433f151 do_mkdirat+0x1 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4bc7a33 do_syscall_64+0x33 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4c0008c entry_SYSCALL_64_after_hwframe+0x44 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
            7f563b57237b __GI___mkdir+0xb (/usr/lib64/libc-2.33.so)
        3d4c4c454853006f [unknown] ([unknown])
```

How About Kprobes?

```
[root@fedora ~]# perf record -e kprobes:* -aRg /bin/sh ./commands.sh
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.331 MB perf.data (2 samples) ]
[root@fedora ~]# perf script
mkdir 93151 [001] 320801.253134: kprobes:myprobenoargs: (ffffffffa433f150)
        ffffffffa433f151 do_mkdirat+0x1 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4bc7a33 do_syscall_64+0x33 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4c0008c entry_SYSCALL_64_after_hwframe+0x44 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
            7fd43c15537b __GI___mkdir+0xb (/usr/lib64/libc-2.33.so)
        3d4c4c454853006f [unknown] ([unknown])

mkdir 93151 [001] 320801.253149:  kprobes:myprobewargs: (ffffffffa433f150) pathname=0x7ffc46f60385 mode=0x1ff
        ffffffffa433f151 do_mkdirat+0x1 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4bc7a33 do_syscall_64+0x33 (/usr/lib/debug/lib/modules/5.11.16-300.fc34.x86_64/vmlinux)
        ffffffffa4c0008c entry_SYSCALL_64_after_hwframe+0x44 (/usr/lib/debug/lib/modules/5.11.16-
300.fc34.x86_64/vmlinux)
            7fd43c15537b __GI___mkdir+0xb (/usr/lib64/libc-2.33.so)
        3d4c4c454853006f [unknown] ([unknown])
```

Multiple Events

```
[root@fedora ~]# perf record -e "{kprobes:*, probe:*}" -aRg /bin/sh
./commands.sh
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.338 MB perf.data (36 samples) ]
[root@fedora ~]# perf script
[...shows all probes and kprobes firing...]
```

```
[root@fedora ~]# cat /sys/kernel/tracing/kprobe_profile
  myprobewargs                                            1091               0
  myprobenoargs                                           1090               0
  do_mkdirat                                                37               0
  do_writepages                                             42               4
```

What Perf can do best: Perf stat

- Trace many kins of events (see the all with “perf list”)

- branch-instructions OR branches [Hardware event]
- L1-dcache-load-misses [Hardware cache event]
- cpu-migrations OR migrations [Software event]
- branch-instructions OR cpu/branch-instructions/ [Kernel PMU event]
- kmem:kmalloc [Tracepoint event]

Statistics with Perf

```
[root@fedora ~]# perf stat /bin/sh commands.sh

 Performance counter stats for '/bin/sh commands.sh':

              3.04 msec task-clock                #    0.311 CPUs utilized
                 6      context-switches          #    0.002 M/sec
                 0      cpu-migrations            #    0.000 K/sec
               451      page-faults               #    0.148 M/sec
         9,956,869      cycles                    #    3.272 GHz
         6,997,743      instructions              #    0.70  insn per cycle
         1,415,962      branches                  #  465.260 M/sec
            48,430      branch-misses             #    3.42% of all branches

       0.009791147 seconds time elapsed

       0.001567000 seconds user
       0.002377000 seconds sys
```

```
[root@fedora ~]# perf stat -e branch-instructions,branch-misses,cycles /bin/sh ./commands.sh
mkdir: cannot create directory ‘./foooooo’: File exists

 Performance counter stats for '/bin/sh ./commands.sh':

         1,427,421      branch-instructions
            49,927      branch-misses             #    3.50% of all branches
         8,738,213      cycles

       0.010067903 seconds time elapsed

       0.000000000 seconds user
       0.004418000 seconds sys


[root@fedora ~]# perf stat -e branch-instructions,branch-misses,cycles -r 3 /bin/sh ./commands.sh
mkdir: cannot create directory ‘./foooooo’: File exists
mkdir: cannot create directory ‘./foooooo’: File exists
mkdir: cannot create directory ‘./foooooo’: File exists

 Performance counter stats for '/bin/sh ./commands.sh' (3 runs):

         1,408,369      branch-instructions                                           ( +-  0.33% )
            50,455      branch-misses             #    3.58% of all branches          ( +-  0.49% )
         8,065,526      cycles                                                        ( +-  1.45% )

           0.01466 +- 0.00110 seconds time elapsed  ( +-  7.48% )
```

BPF

- Infrastructure that allows user defined programs to execute in kernel space.
- Programs written in C and translated into BPF instructions using compiler (gcc or clang/llvm), loaded in kernel and executed
- 10 64-bit registers
- Language with ~100 instructions (including “bpf_call” for calling helper kernel functions from BPF programs)
- Safety checks are performed by BPF program verifier in kernel
- Kernel has JITs for several architectures
- Due to its history, you will find references to cBPF (classic), eBPF (extended), now simply called BPF
- Needs a userspace program to do the housekeeping: compile the bpf program, load it, etc

BPF Programs

- Different types of programs. Type determines how to interpret the context argument (mainly). Correspond to areas of BPF use in kernel
  - BPF_PROG_TYPE_SOCKET_FILTER
  - BPF_PROG_TYPE_SCHED_CLS
  - BPF_PROG_TYPE_SCHED_ACT
  - BPF_PROG_TYPE_XDP
  - BPF_PROG_TYPE_KPROBE
  - BPF_PROG_TYPE_TRACEPOINT
  - BPF_PROG_TYPE_PERF_EVENT
  - [….]
- BPF_PROG_RUN(ctx, prog): kernel macro that executes the program instructions. Has 2 arguments: pointer to context, array of bpf program instructions

Some BPF Concepts

- Each BPF program is run within a context (ctx argument)
- Context may be used when calling helper functions, as their first argument
- Context provides data on which the BPF program operates:
  - (k)probes: it is the register set
  - Tracepoints: it is the format string
  - Networking filters: it is the socket buffer

- A BPF program can call certain helper functions.
- Helper Functions must be known: enum bpf_func_id values in include/uapi/linux/bpf.h
  - Map operations
  - Tracing
  - Networking
  - […]

Maps

- A map is a key-value store
- Transfer data from BPF programs to userspace or to kernel or vice versa; share data among many BPF programs
- A map is identified by a file descriptor returned by a bpf() system call in a userspace program that creates the map
- Attributes of a map: max elements, size of key, size of value
- Many types of maps: BPF_MAP_TYPE_ARRAY, BPF_MAP_TYPE_HASH, BPF_MAP_TYPE_PROG_ARRAY, BPF_MAP_TYPE_PERF_EVENT_ARRAY, BPF_MAP_TYPE_STACK_TRACE, BPF_MAP_TYPE_CGROUP_ARRAY,….
- Maps operations (only specific ones allowed):
  - by user level programs (via bpf() syscall) or
  - by BPF programs via helper functions
- To close a map, call close() on the descriptor
- Maps (and BPF) can be persistent across termination of the process that created the map

How to Use it?

- Gnarly!
- For all the gory details see old presentation: https://events.linuxfoundation.org/sites/events/files/slides/tracing-linux-ezannoni-linuxcon-ja-2015_0.pdf
- Some tools to the rescue
- BCC: BPF Compiler Collection
  - Set of many programs to perform common tracing and performance anylisys tasks
  - Not specifically tied to tracing, but generic for BPF usage
  - https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
  - Uses llvm/clang library to create BPF maps, resolve relocations, load an verify BPF programs in the kernel. Python scripts.
  - You can use API to write new scripts

Bcc Script Example

Trace new processes via exec() syscalls.

```
[root@fedora ~]# /usr/share/bcc/tools/execsnoop
[...some llvm warnings...]
3 warnings generated.
PCOMM            PID    PPID   RET ARGS
sed              137934 137932   0 /usr/bin/sed s/^ *[0-9]\+ *//
vte-urlencode-c  137935 68989    0 /usr/libexec/vte-urlencode-cwd
ls               137936 68989    0 /usr/bin/ls --color=auto
sed              137939 137937   0 /usr/bin/sed s/^ *[0-9]\+ *//
vte-urlencode-c  137940 68989    0 /usr/libexec/vte-urlencode-cwd
systemd-userwor  137941 965      0
setroubleshootd  137942 1        0 /usr/sbin/setroubleshootd -f
rpm              137943 137942   0
rpm              137945 137942   0
uname            137946 137942   0
rpm              137947 137942   0
SetroubleshootP  137950 1        0 /usr/share/setroubleshoot/SetroubleshootPrivileged.py
rpm              137953 137950   0
^C
[root@fedora ~]# wc -l  /usr/share/bcc/tools/execsnoop
307 /usr/share/bcc/tools/execsnoop
```

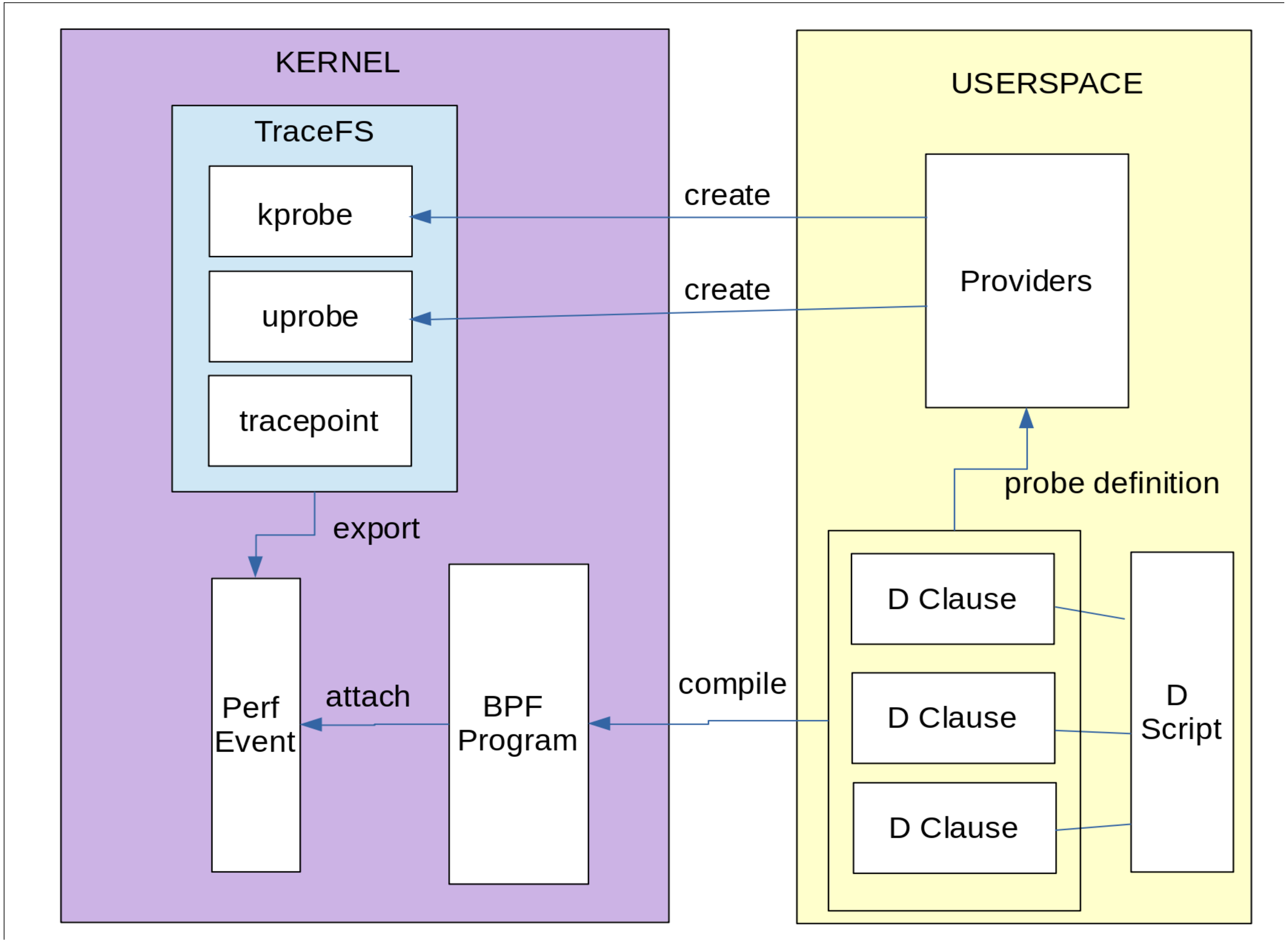
DTrace

- Well documented feature set
- Available on multiple operating systems
- Powerful programmable tracing system
- Easy enough to do very basic tracing
- Powerful enough for complex tracing across many probes
- Stable enough for long-term tracing (incl. Always-on tracing)
- DTrace on Linux first version in Oct 2011
- Under active development ever since
- Now Re-implement without big kernel patches
- Leverage BPF and other kernel facilities
- https://github.com/oracle/dtrace-utils
- https://oss.oracle.com/pipermail/dtrace-devel/

Rearchitecting DTrace

- Implement as much as possible in Userspace, greatly limit need for kernel changes
- Kernel provides probing mechanisms
- BPF gives us an execution engine
- BPF programs attach to probes
- Output written to perf_event ring buffer
- Each D clause is compiled into a BPF function dt_func(dt_dctx_t *dctx)
- BPF trampoline program generated for each probe that is being enabled
- Trampoline calls the BPF functions for the probe clauses

A Simplified Dtrace Diagram



Simple Dtrace Example

```
/* tick.d -- Perform action at */
/* regular intervals */

BEGIN
{
  i = 0;
}

profile:::tick-1sec
{
  printf("i = %d\n",++i);
}

END
{
  trace(i);
}
```

```
[opc@elena-x86-20210418 ~]$ sudo dtrace -s tick.d
DTrace 2.0.0 [Pre-Release with limited functionality]
dtrace: script 'tick.d' matched 3 probes
CPU     ID                     FUNCTION:NAME
  1 107384                        :tick-1sec i = 1

  1 107384                        :tick-1sec i = 2

  1 107384                        :tick-1sec i = 3

  1 107384                        :tick-1sec i = 4

  1 107384                        :tick-1sec i = 5

  1 107384                        :tick-1sec i = 6

^C
  1      2                              :END         6
```

...behind the scenes

```
[opc@elena-x86-20210418 ~]$ sudo cat /sys/kernel/debug/tracing/uprobe_events
p:dt_1803501_dtrace/BEGIN /usr/lib64/libdtrace.so.2.0.0:0x0000000000091b90
p:dt_1803501_dtrace/END /usr/lib64/libdtrace.so.2.0.0:0x0000000000091ba0
```

```
 [opc@elena-x86-20210418 ~]$ sudo bpftool prog
 109: kprobe  tag a0a7f781a0ffd0ad  gpl
         loaded_at 2021-05-05T01:33:17+0000  uid 0
         xlated 1096B  jited 680B  memlock 4096B  map_ids 225,228,230
 110: kprobe  tag 3e2573ffe8b60d7d  gpl
         loaded_at 2021-05-05T01:33:17+0000  uid 0
         xlated 1184B  jited 726B  memlock 4096B  map_ids 225,228,230,226
 111: perf_event  tag 074497c02e965b39  gpl
         loaded_at 2021-05-05T01:33:17+0000  uid 0
         xlated 784B  jited 542B  memlock 4096B  map_ids 225,228,230,226
```

...under the hood

```
[opc@elena-x86-20210418 ~]$ sudo dtrace -xdisasm=8 -S -s tick.d
DTrace 2.0.0 [Pre-Release with limited functionality]
dtrace: script 'tick.d' matched 3 probes

Disassembly of final program dtrace:::BEGIN:
INS OFF    OPCODE                       INSTRUCTION
000 0000: bf 8 1 0000 00000000    mov   %r8, %r1
001 0008: 7b a 8 ffc8 00000000    stdw [%fp-56], %r8
002 0016: 62 a 0 ffd0 00000000    stw  [%fp-48], 0
[...]

Disassembly of final program dtrace:::END:
INS OFF    OPCODE                       INSTRUCTION
000 0000: bf 8 1 0000 00000000    mov   %r8, %r1
001 0008: 7b a 8 ffc8 00000000    stdw [%fp-56], %r8
002 0016: 62 a 0 ffd0 00000000    stw  [%fp-48], 0
[...]

Disassembly of final program profile:::tick-1sec:
INS OFF    OPCODE                       INSTRUCTION
000 0000: bf 8 1 0000 00000000    mov   %r8, %r1
001 0008: 7b a 8 ffc8 00000000    stdw [%fp-56], %r8
002 0016: 62 a 0 ffd0 00000000    stw  [%fp-48], 0
[...]
```

Other Examples

FBT creates kprobes underneath:

```
[opc@elena-x86-20210418 ~]$ sudo dtrace -q -n fbt::__kmalloc:entry'{ @ = count(); }'
DTrace 2.0.0 [Pre-Release with limited functionality]
^C
          5893213

[opc@elena-x86-20210418 ~]$ sudo cat /sys/kernel/debug/tracing/kprobe_events
p:dt_1804825_fbt_entry/__kmalloc __kmalloc
```

Predicate and multiple clauses:

```
[opc@elena-x86-20210418 ~]$ sudo dtrace -n __kmalloc:entry'{ printf("%x %x\n", arg1, arg1
& 0x200); }' -n __kmalloc:entry'/arg1 & 0x200/ { printf("Found one!\n"); }'
```

A Complex Example: histogram and timing of syscalls

```
#pragma D option quiet

syscall:::entry
/ progenyof($target) /
{
  self->time = timestamp;
  @maxbytes[probefunc] = max(arg2);
}
syscall:::return
/self->time != 0 && progenyof($target) /
{
  @calls[probefunc] = count();
  @elapsed[probefunc] = sum(timestamp - self->time);
  @stdelapsed[probefunc, errno] = stddev(timestamp - self->time);
  @quantelapsed[probefunc, errno] = quantize(timestamp - self->time);
}
END
{
  trace ("\nNum calls:\n");
  printa(@calls);
  trace("\n\nElapsed time:\n");
  printa(@elapsed);
  trace("\n\nStd dev of elapsed time by errno\n");
  printa(@stdelapsed);
  trace("\n\nHistogram elapsed time by errno\n");
  printa(@quantelapsed);
  trace ("\n\nMax bytes:\n");
  printa(@maxbytes);
}
```

```
[nix-test~]$ dtrace -o foo -s timings-hist.d
```

Output part 1

```
Num calls:

  accept4                                                          1
  [...]
  getpgrp                                                          3
  getppid                                                          3
  nanosleep                                                        3
  vfork                                                            3
  waitid                                                           3
  [...]
  connect                                                         25
  [...]
  ptrace                                                         172
  [...]
  close                                                        24675

Elapsed time:

  gettid                                                            4753
  dup                                                               4998
  dup3                                                              5098
  getpriority                                                       5125
  [...]
  connect                                                        1349306
  [...]
```

Output part 2

```
Std dev of elapsed time by errno

  connect                                                  111                0
  connect                                                    2            47669
  connect                                                    0           129731

Histogram elapsed time by errno

connect                                                  111
           value  ------------- Distribution ------------- count
            4096 |                                         0
            8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1
           16384 |                                         0

connect                                                    2
           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |@@@@@@@@                                 4
            8192 |@@@@@@@@@@@@@@@@@@                       9
           16384 |@@@@@@@@@@@@                             6
           32768 |                                         0
           65536 |                                         0
          131072 |@@                                       1
          262144 |                                         0

connect                                                    0
           value  ------------- Distribution ------------- count
           32768 |                                         0
           65536 |@@@@@@@@@@@@@@@@@@@@                     2
          131072 |@@@@@@@@@@                               1
          262144 |@@@@@@@@@@                               1
          524288 |                                         0
```

Output Part 3

```
Max bytes:

  clone                                                       0
  dup3                                                        0
  ftruncate                                                   0
  mknod                                                       0
  prctl                                                       0
  setgid                                                      0
  setpriority                                                 0
  setsid                                                      0
[...]
  connect                                                   110
[...]
```

Another little example

```
$ dtrace -n ‘syscall:::entry {@num[execname] = count();}’
dtrace: description 'syscall:::entry ' matched 319 probes


  lsmd                                                              4
  sudo                                                              7
  dbus-daemon                                                      20
  gmain                                                            24
  gdbus                                                            58
  in:imjournal                                                     84
  tuned                                                           125
  NetworkManager                                                  128
  irqbalance                                                      222
  systemd                                                         360
  dtrace                                                          979
```

BpfTrace

- Provides a collection of scripts that can do tracing using bcc under the hood.
- Wrapper around BCC, provides higher level syntax
- Similar syntax to DTrace
- Uses BPF, of course

BpfTrace example

```
[root@fedora ~]# bpftrace -e 'tracepoint:raw_syscalls:sys_enter
{@[comm]=count();}'
Attaching 1 probe...
^C

@[sedispatch]: 1
@[goa-identity-se]: 2
@[gsd-sharing]: 2
@[gsd-media-keys]: 4
@[seapplet]: 4
@[gsd-wacom]: 4
@[gsd-xsettings]: 4
@[Privileged Cont]: 4
@[ibus-extension-]: 4
@[ibus-x11]: 4
[...]
```

Other Tools

- Trace-cmd: a front end for ftrace. User space tool, many options, very flexible. Works with Kernelshark: GUI on top of trace-cmd Available in https://git.kernel.org/pub/scm/utils/trace-cmd
- Systemtap: https://sourceware.org/systemtap/
- LTTng: https://lttng.org/