Using trace event

October, 2017, Tokyo

Joey Lee

SUSE Labs Taipei

Agenda

- Tracepoint and Trace event
- Review sched_switch
  - Following Steven Rostedt’s “Using the TRACE_EVENT() macro” articles on LWN.net
- sillymod kernel module
- Q&A

Tracepoint and Trace event

Steven Rostedt

- Creator and maintainer of ftrace.
- Steven Rostedt’s “Using the TRACE_EVENT() macro” articles on LWN.net
  - Part 1: https://lwn.net/Articles/379903/
  - Part 2: https://lwn.net/Articles/381064/
  - Part 3: https://lwn.net/Articles/383362/

Tracepoints

- v2.6.28-rc1 (2008)
- A tracepoint placed in code provides a hook to call a function (probe) that you can provide at runtime.
- A tracepoint can be "on" (a probe is connected to it) or "off" (no probe is attached).
- When a tracepoint is "on", the function you provide is called each time the tracepoint is executed, in the execution context of the caller. [1]

Purpose of tracepoints

- Tracepoints can be used without creating custom kernel modules to register probe functions using the event tracing infrastructure.
- Simplistically, tracepoints represent important events that can be taken in conjunction with other tracepoints to build a "Big Picture" of what is going on within the system. [2]
- Unlike the Ftrace function tracer, a tracepoint can record more than just the function being entered. A tracepoint can record local variables of the function. [3]

TRACE_EVENT

- The developer need not understand how Ftrace works, they only need to create their tracepoint using the TRACE_EVENT() macro. [3]
- Another objective of the design of the TRACE_EVENT() macro was to not couple it to Ftrace or any other tracer. It is agnostic to the tracers that use it, which is apparent now that TRACE_EVENT() is also used by perf, LTTng and SystemTap. [3]

Review sched_switch

sched_switch trace event (name)

- name - the name of the tracepoint to be created. [3]

  /*

   * Tracepoint for task switches, performed by the scheduler:

   */

  TRACE_EVENT(sched_switch,

sched_switch trace event (prototype)

- prototype - the prototype for the tracepoint callbacks

```
        TP_PROTO(bool preempt,
                 struct task_struct *prev,
                 struct task_struct *next),
```

- **trace_sched_switch(bool preempt, struct task_struct *prev, struct task_struct *next);**

sched_switch trace event (arguments)

- args - the arguments that match the prototype.

  TP_ARGS(preempt, prev, next),

- The tracepoint code, when activated, will call the callback functions (more than one callback may be assigned to a given tracepoint). The macro that creates the tracepoint must have access to both the prototype and the arguments. [3]

```
#define TRACE_POINT(name, proto, args) \
    void trace_##name(proto)          \
    {                                 \
        if (trace_##name##_active)  \
            callback(args);     \
    }
```

sched_switch trace event (struct)

- struct - the structure that a tracer could use (but is not required to) to store the data passed into the tracepoint. [3]
- This parameter describes the structure layout of the data that will be stored in the tracer's ring buffer. [3]

```
TP_STRUCT__entry(
        __array(     char,   prev_comm,      TASK_COMM_LEN   )
        __field(     pid_t,  prev_pid                        )
        __field(     int,    prev_prio                       )
        __field(     long,   prev_state                      )
        __array(     char,   next_comm,      TASK_COMM_LEN   )
        __field(     pid_t,  next_pid                        )
        __field(     int,    next_prio                       )
),
```

sched_switch trace event (struct) (cont.)

```
struct {
  char   prev_comm[TASK_COMM_LEN];
  pid_t  prev_pid;
  int    prev_prio;
  long   prev_state;
  char   next_comm[TASK_COMM_LEN];
  pid_t  next_pid;
  int    next_prio;
};
```

sched_switch trace event (assign)

- assign - the C-like way to assign the data to the structure. [3]

```
TP_fast_assign(
        memcpy(__entry->next_comm, next->comm, TASK_COMM_LEN);
        __entry->prev_pid       = prev->pid;
        __entry->prev_prio      = prev->prio;
        __entry->prev_state     = __trace_sched_switch_state(preempt, prev);
        memcpy(__entry->prev_comm, prev->comm, TASK_COMM_LEN);
        __entry->next_pid       = next->pid;
        __entry->next_prio      = next->prio;
        /* XXX SCHED_DEADLINE */
),
```

sched_switch trace event (print)

- print - the way to output the structure in human readable ASCII format.

```
        TP_printk("prev_comm=%s prev_pid=%d prev_prio=%d prev_state=%s
%s ==> next_comm=%s next_pid=%d next_prio=%d",
                __entry->prev_comm, __entry->prev_pid, __entry->prev_prio,
                __entry->prev_state & (TASK_STATE_MAX-1) ?
                  __print_flags(__entry->prev_state & (TASK_STATE_MAX-1), "|",
                        { 1, "S"} , { 2, "D" }, { 4, "T" }, { 8, "t" },
                        { 16, "Z" }, { 32, "X" }, { 64, "x" },
                        { 128, "K" }, { 256, "W" }, { 512, "P" },
                        { 1024, "N" }) : "R",
                __entry->prev_state & TASK_STATE_MAX ? "+" : "",
                __entry->next_comm, __entry->next_pid, __entry->next_prio)
```

Header file for sched_switch

- include/trace/events/sched.h

```
#undef TRACE_SYSTEM
#define TRACE_SYSTEM sched

#if !defined(_TRACE_SCHED_H) || defined(TRACE_HEADER_MULTI_READ)
#define _TRACE_SCHED_H

#include <linux/sched/numa_balancing.h>
#include <linux/tracepoint.h>
```

- The TRACE_SYSTEM defines what group the TRACE_EVENT() macros in the file belong to. [3]
- The TRACE_HEADER_MULTI_READ test allows this file to be included more than once. [3]
- The tracepoint.h file is required for TRACE_EVENT() marco.

Header file for sched_switch (cont.)

- include/trace/events/sched.h

  ```
  #endif /* _TRACE_SCHED_H */


  /* This part must be outside protection */
  #include <trace/define_trace.h>
  ```

- The define_trace.h is where all the magic lies in creating the tracepoints. …this file must be included at the bottom of the trace header file outside the protection of the #endif. [3]

Using the tracepoint

- kernel/sched/core.c

```
[...snip]
#include "../smpboot.h"

#define CREATE_TRACE_POINTS
#include <trace/events/sched.h>
[...snip]
static void __sched notrace __schedule(bool preempt)
{
[...snip]
                ++*switch_count;

                trace_sched_switch(preempt, prev, next);

                /* Also unlocks the rq: */
                rq = context_switch(rq, prev, next, &rf);
[...snip]
```

- To use the tracepoint, the trace header must be included, but one C file (and only one) must also define CREATE_TRACE_POINTS before including the trace. [3]

V4.12 mainline kernel

Enable sched_switch event

- cd /sys/kernel/debug/tracing

  ```
  # echo 1 > events/sched/sched_switch/enable
  ```

  or

  ```
  # echo sched_switch > set_event
  ```

-
  ```
  # cat trace_pipe
  [...snip]
  sshd-2926  [000] d... 97823.734835: sched_switch:
  prev_comm=sshd prev_pid=2926 prev_prio=120
  prev_state=S ==> next_comm=kworker/u9:1
  next_pid=3933 next_prio=120
  ...
  ```

Size of text section

-
  ```
  text      data   bss   dec      hex   filename
  452114    2788   3520  458422   6feb6 fs/xfs/xfs.o.notrace
  996954    38116  4480  1039550  fdcbe fs/xfs/xfs.o.trace
  638482    38116  3744  680342   a6196 fs/xfs/xfs.o.class
  ```
- enabling the trace events causes the xfs.o text section to double in size! [4]
- If two events have the same TP_PROTO, TP_ARGS and TP_STRUCT__entry, there should be a way to have these events share the functions that they use. [4]

DECLARE_EVENT_CLASS

- include/trace/events/sched.h

```
/*
 * Tracepoint for waking up a task:
 */
DECLARE_EVENT_CLASS(sched_wakeup_template,

        TP_PROTO(struct task_struct *p),

        TP_ARGS(__perf_task(p)),

        TP_STRUCT__entry( [...snip]

        TP_fast_assign( [...snip]

        TP_printk("comm=%s pid=%d prio=%d target_cpu=%03d", [...snip]
);
```

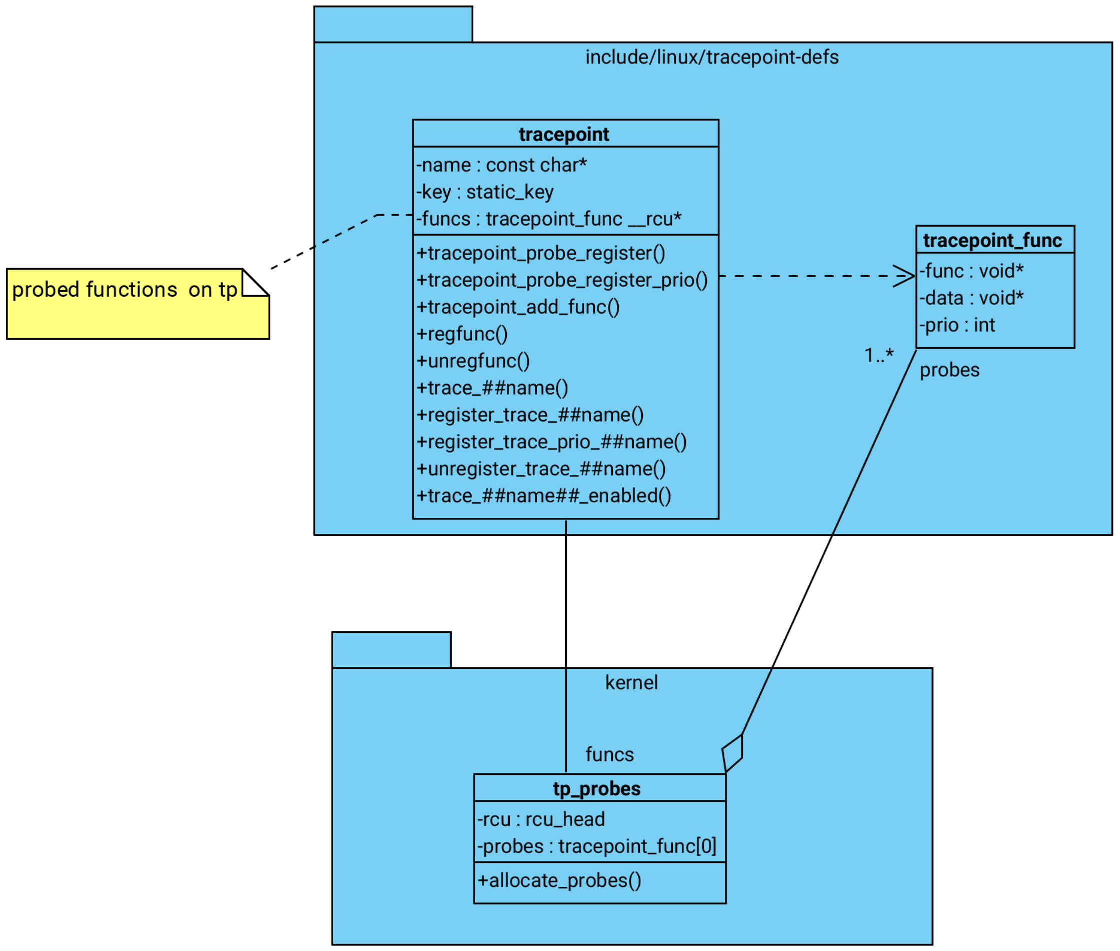
- The DECLARE_EVENT_CLASS() macro has the exact same format as TRACE_EVENT()

DEFINE_EVENT

- include/trace/events/sched.h

```
DEFINE_EVENT(sched_wakeup_template, sched_waking,
             TP_PROTO(struct task_struct *p),
             TP_ARGS(p));
DEFINE_EVENT(sched_wakeup_template, sched_wakeup,
             TP_PROTO(struct task_struct *p),
             TP_ARGS(p));
DEFINE_EVENT(sched_wakeup_template,
sched_wakeup_new,
             TP_PROTO(struct task_struct *p),
             TP_ARGS(p));
```

Tracepoint conceptual model

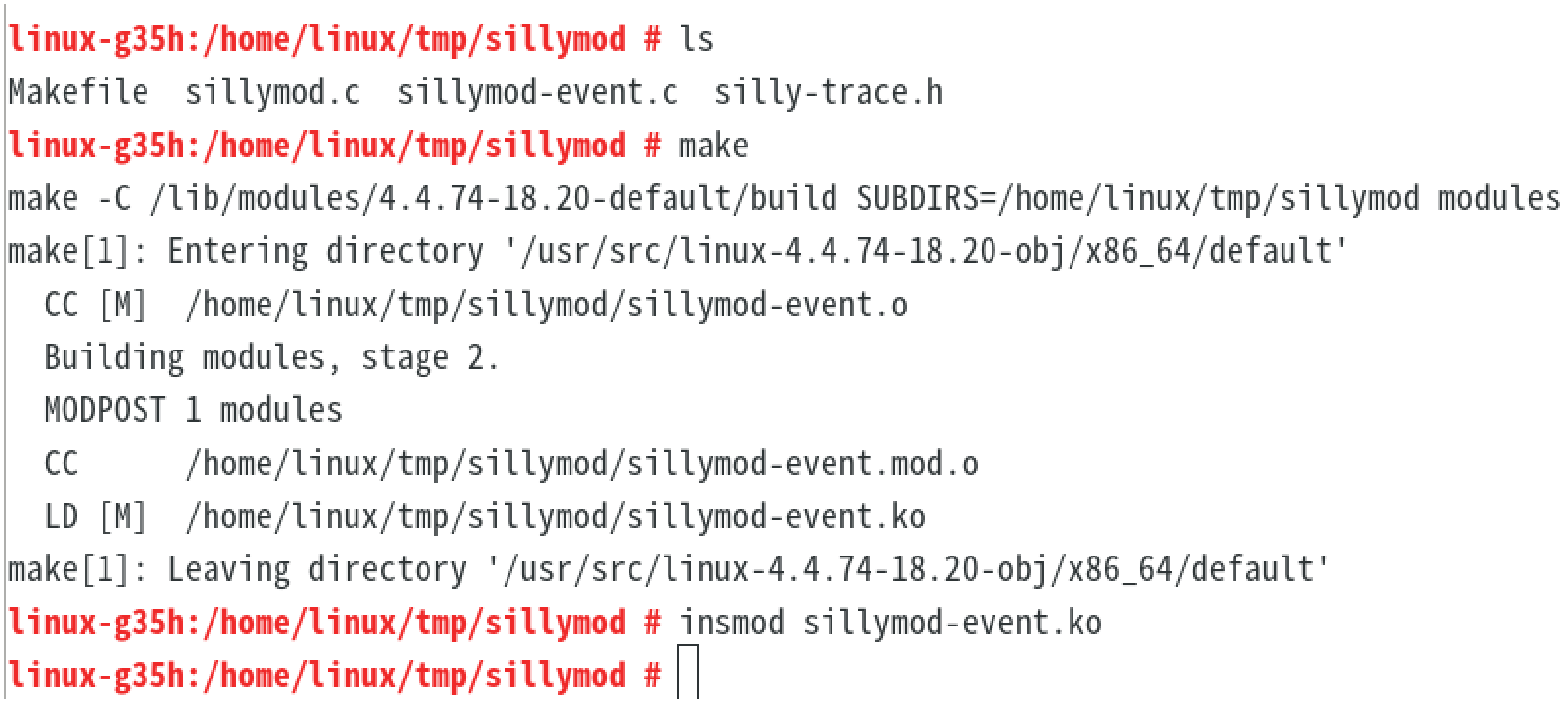


sillymod

sillymode

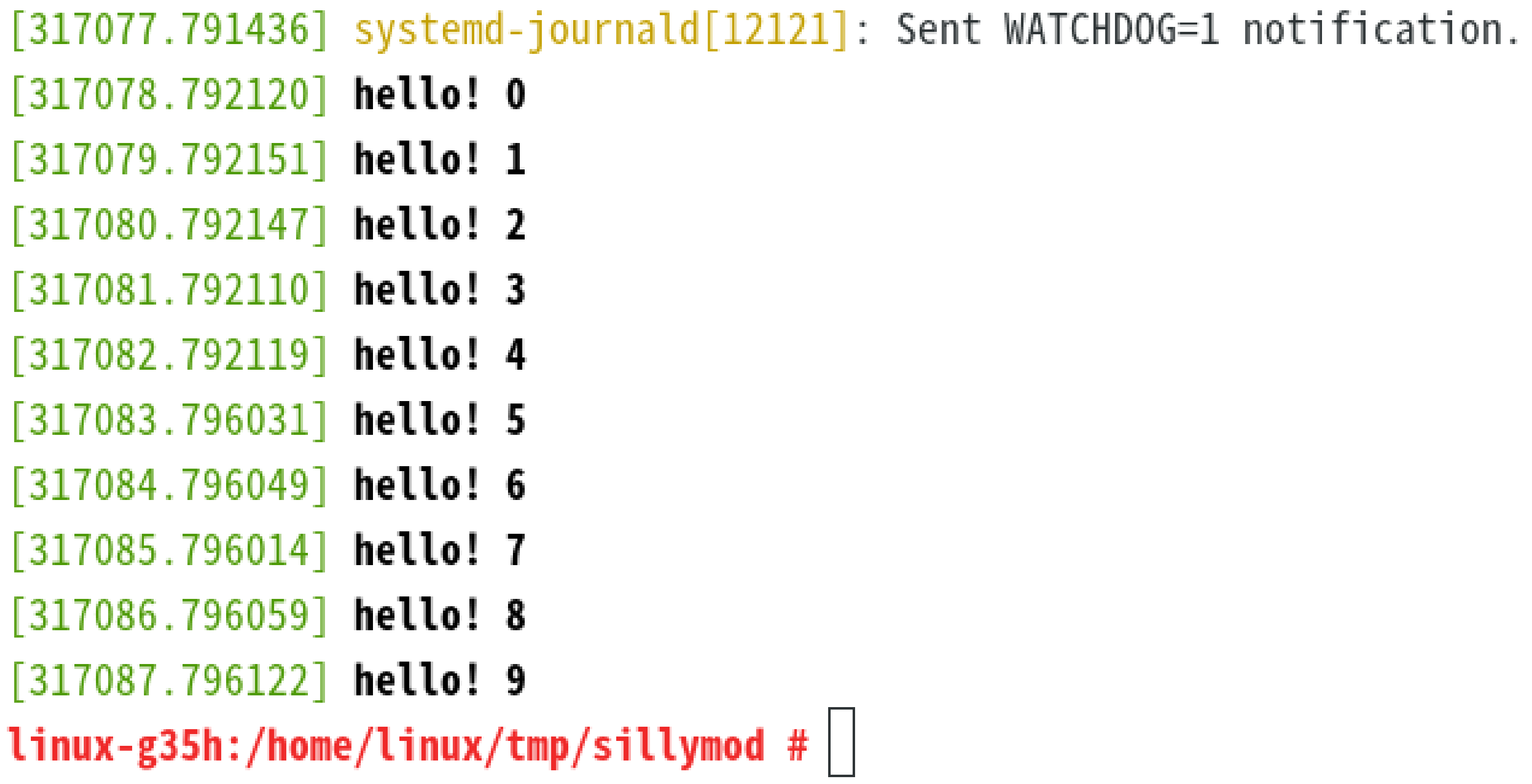
- sillymod.c
  - Original kernel module for referenece
- sillymod-event.c
  - Kernel module with me_silly trace event
- silly-trace.h
  - Defined me_silly trace event by TRACE_EVET marco
- Makefile
  - Build sillymod-event.ko

Build sillymod-event.ko



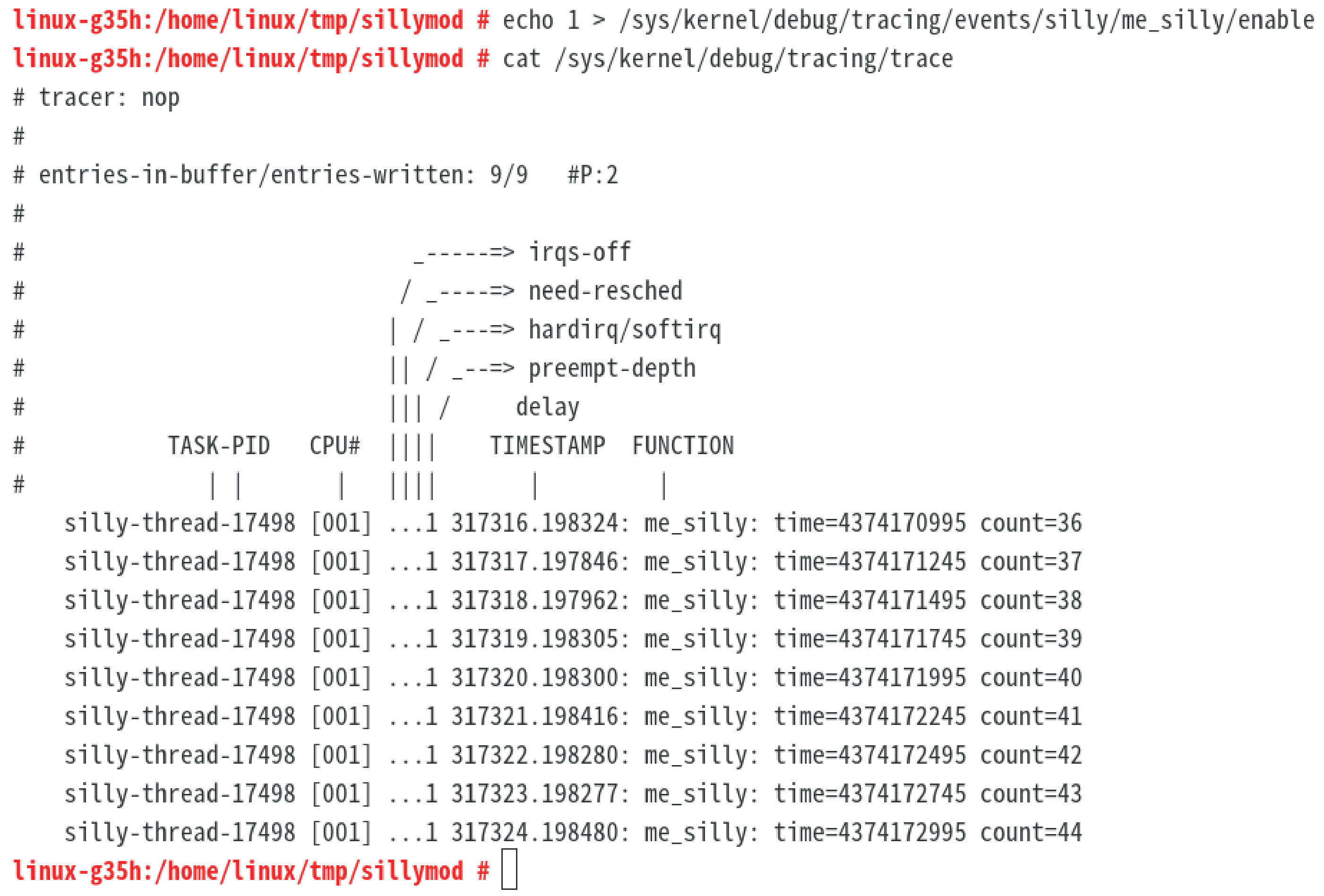
```
linux-g35h:/home/linux/tmp/sillymod # ls
Makefile  sillymod.c  sillymod-event.c  silly-trace.h
linux-g35h:/home/linux/tmp/sillymod # make
make -C /lib/modules/4.4.74-18.20-default/build SUBDIRS=/home/linux/tmp/sillymod modules
make[1]: Entering directory '/usr/src/linux-4.4.74-18.20-obj/x86_64/default'
  CC [M]  /home/linux/tmp/sillymod/sillymod-event.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/linux/tmp/sillymod/sillymod-event.mod.o
  LD [M]  /home/linux/tmp/sillymod/sillymod-event.ko
make[1]: Leaving directory '/usr/src/linux-4.4.74-18.20-obj/x86_64/default'
linux-g35h:/home/linux/tmp/sillymod # insmod sillymod-event.ko
linux-g35h:/home/linux/tmp/sillymod # 
```

dmesg



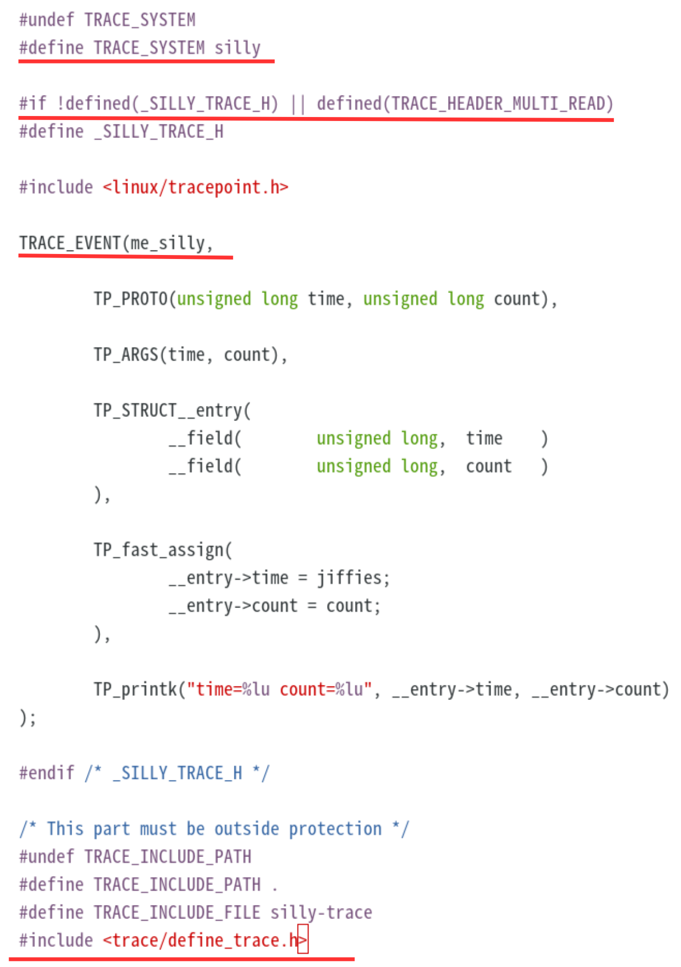
```
[317077.791436] systemd-journald[12121]: Sent WATCHDOG=1 notification.
[317078.792120] hello! 0
[317079.792151] hello! 1
[317080.792147] hello! 2
[317081.792110] hello! 3
[317082.792119] hello! 4
[317083.796031] hello! 5
[317084.796049] hello! 6
[317085.796014] hello! 7
[317086.796059] hello! 8
[317087.796122] hello! 9
linux-g35h:/home/linux/tmp/sillymod #
```

Enable me_silly event



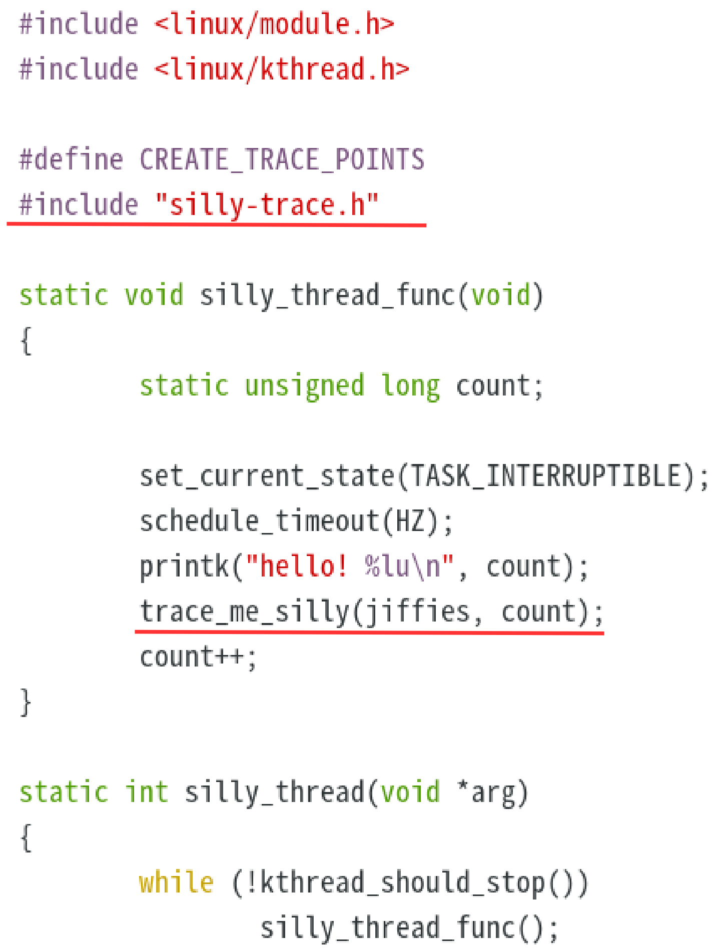
```
linux-g35h:/home/linux/tmp/sillymod # echo 1 > /sys/kernel/debug/tracing/events/silly/me_silly/enable
linux-g35h:/home/linux/tmp/sillymod # cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
# entries-in-buffer/entries-written: 9/9   #P:2
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
    silly-thread-17498 [001] ...1 317316.198324: me_silly: time=4374170995 count=36
    silly-thread-17498 [001] ...1 317317.197846: me_silly: time=4374171245 count=37
    silly-thread-17498 [001] ...1 317318.197962: me_silly: time=4374171495 count=38
    silly-thread-17498 [001] ...1 317319.198305: me_silly: time=4374171745 count=39
    silly-thread-17498 [001] ...1 317320.198300: me_silly: time=4374171995 count=40
    silly-thread-17498 [001] ...1 317321.198416: me_silly: time=4374172245 count=41
    silly-thread-17498 [001] ...1 317322.198280: me_silly: time=4374172495 count=42
    silly-thread-17498 [001] ...1 317323.198277: me_silly: time=4374172745 count=43
    silly-thread-17498 [001] ...1 317324.198480: me_silly: time=4374172995 count=44
linux-g35h:/home/linux/tmp/sillymod # 
```

silly-trace.h



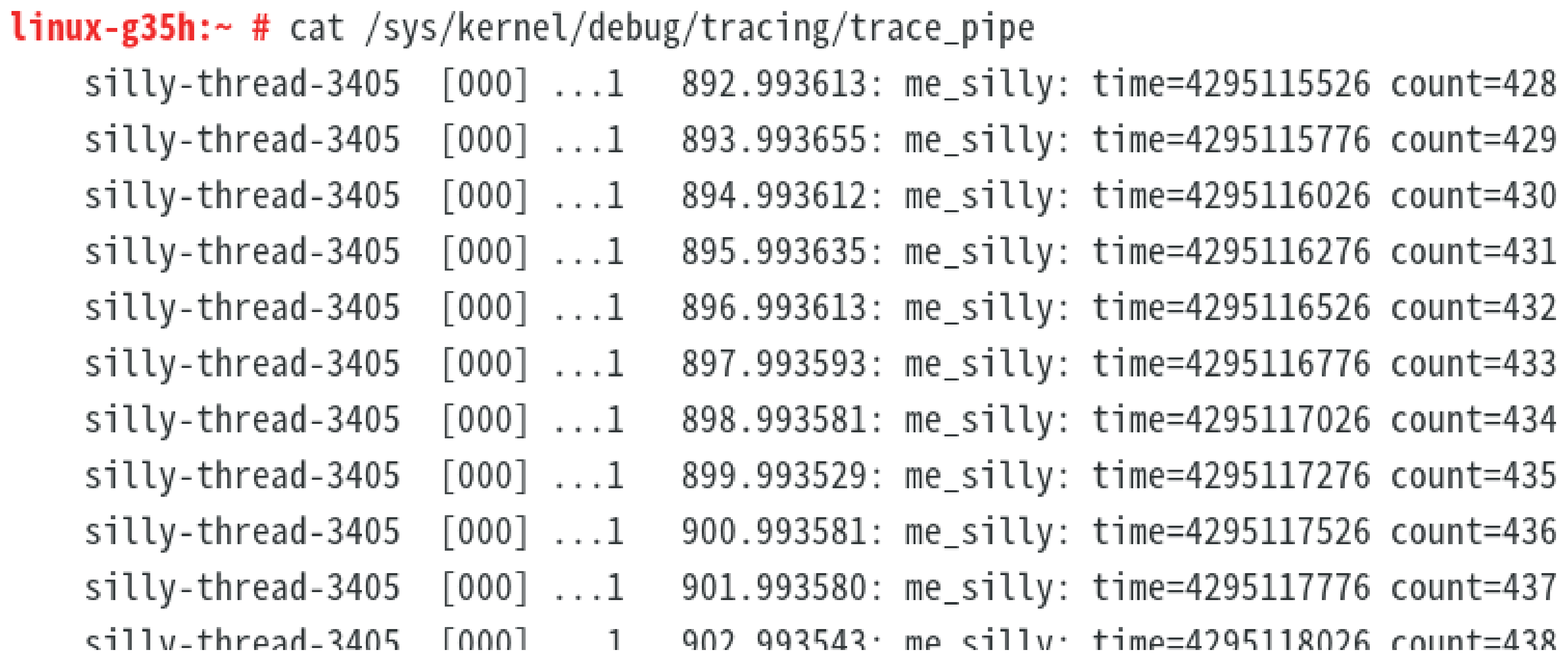
```
#undef TRACE_SYSTEM
#define TRACE_SYSTEM silly

#if !defined(_SILLY_TRACE_H) || defined(TRACE_HEADER_MULTI_READ)
#define _SILLY_TRACE_H

#include <linux/tracepoint.h>

TRACE_EVENT(me_silly,

        TP_PROTO(unsigned long time, unsigned long count),

        TP_ARGS(time, count),

        TP_STRUCT__entry(
                __field(        unsigned long,  time    )
                __field(        unsigned long,  count   )
        ),

        TP_fast_assign(
                __entry->time = jiffies;
                __entry->count = count;
        ),

        TP_printk("time=%lu count=%lu", __entry->time, __entry->count)
);

#endif /* _SILLY_TRACE_H */

/* This part must be outside protection */
#undef TRACE_INCLUDE_PATH
#define TRACE_INCLUDE_PATH .
#define TRACE_INCLUDE_FILE silly-trace
#include <trace/define_trace.h>
```

sillymod-event.c



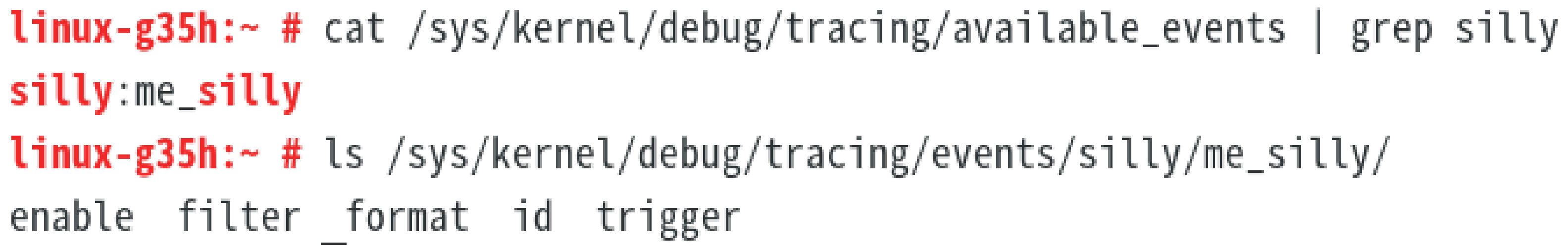
```c
#include <linux/module.h>
#include <linux/kthread.h>

#define CREATE_TRACE_POINTS
#include "silly-trace.h"

static void silly_thread_func(void)
{
        static unsigned long count;

        set_current_state(TASK_INTERRUPTIBLE);
        schedule_timeout(HZ);
        printk("hello! %lu\n", count);
        trace_me_silly(jiffies, count);
        count++;
}

static int silly_thread(void *arg)
{
        while (!kthread_should_stop())
                silly_thread_func();
```

Using trace_pipe



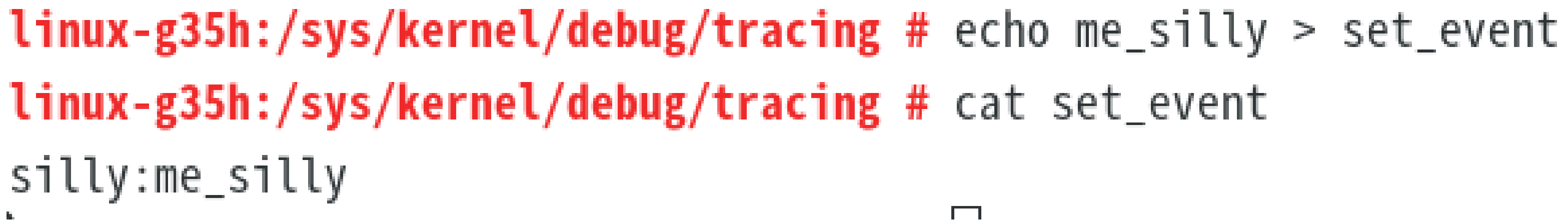
```
linux-g35h:~ # cat /sys/kernel/debug/tracing/trace_pipe
    silly-thread-3405  [000] ...1   892.993613: me_silly: time=4295115526 count=428
    silly-thread-3405  [000] ...1   893.993655: me_silly: time=4295115776 count=429
    silly-thread-3405  [000] ...1   894.993612: me_silly: time=4295116026 count=430
    silly-thread-3405  [000] ...1   895.993635: me_silly: time=4295116276 count=431
    silly-thread-3405  [000] ...1   896.993613: me_silly: time=4295116526 count=432
    silly-thread-3405  [000] ...1   897.993593: me_silly: time=4295116776 count=433
    silly-thread-3405  [000] ...1   898.993581: me_silly: time=4295117026 count=434
    silly-thread-3405  [000] ...1   899.993529: me_silly: time=4295117276 count=435
    silly-thread-3405  [000] ...1   900.993581: me_silly: time=4295117526 count=436
    silly-thread-3405  [000] ...1   901.993580: me_silly: time=4295117776 count=437
    silly-thread-3405  [000]    1   902.993543: me_silly: time=4295118026 count=438
```

Search available_event



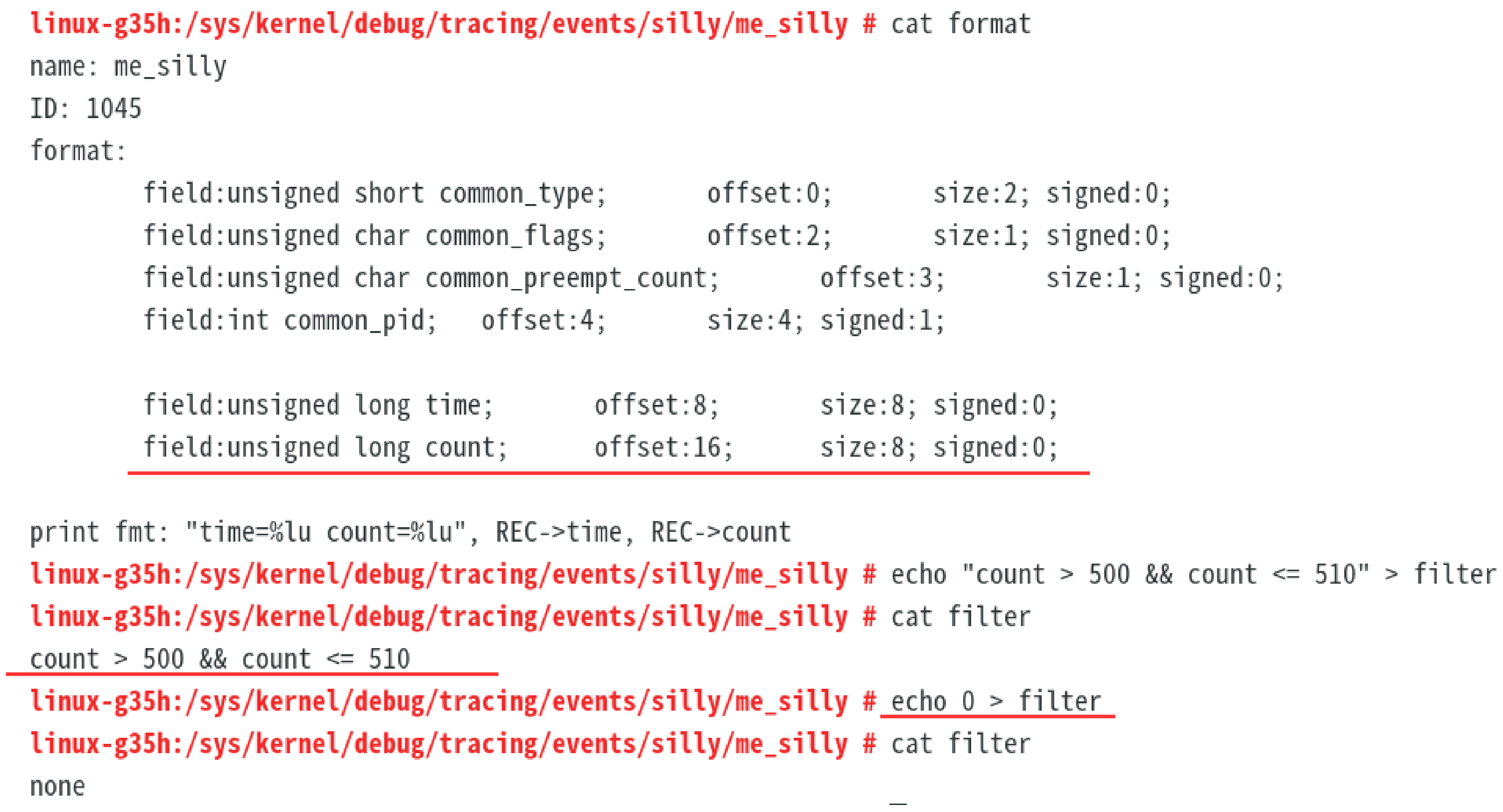
```
linux-g35h:~ # cat /sys/kernel/debug/tracing/available_events | grep silly
silly:me_silly
linux-g35h:~ # ls /sys/kernel/debug/tracing/events/silly/me_silly/
enable  filter  format  id  trigger
```

Using set_event to enable event



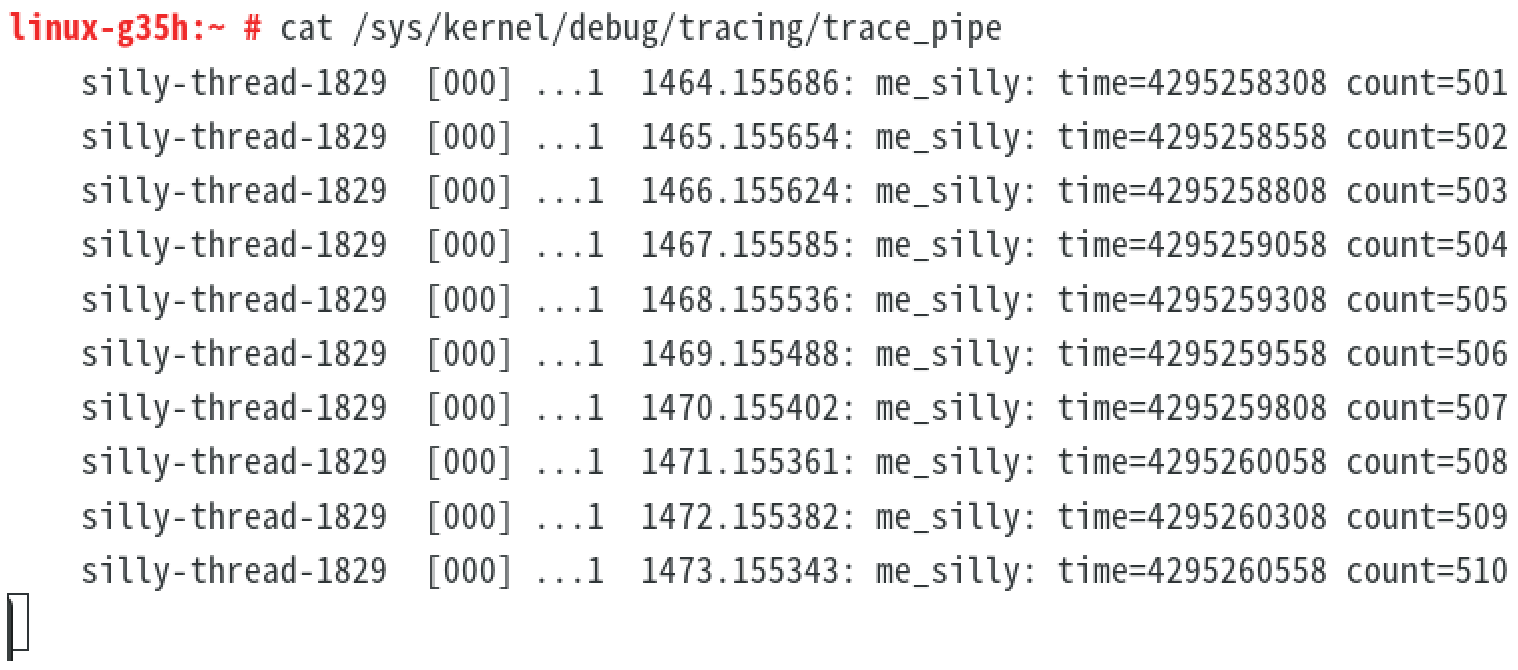
```
linux-g35h:/sys/kernel/debug/tracing # echo me_silly > set_event
linux-g35h:/sys/kernel/debug/tracing # cat set_event
silly:me_silly
```

Setting event filter



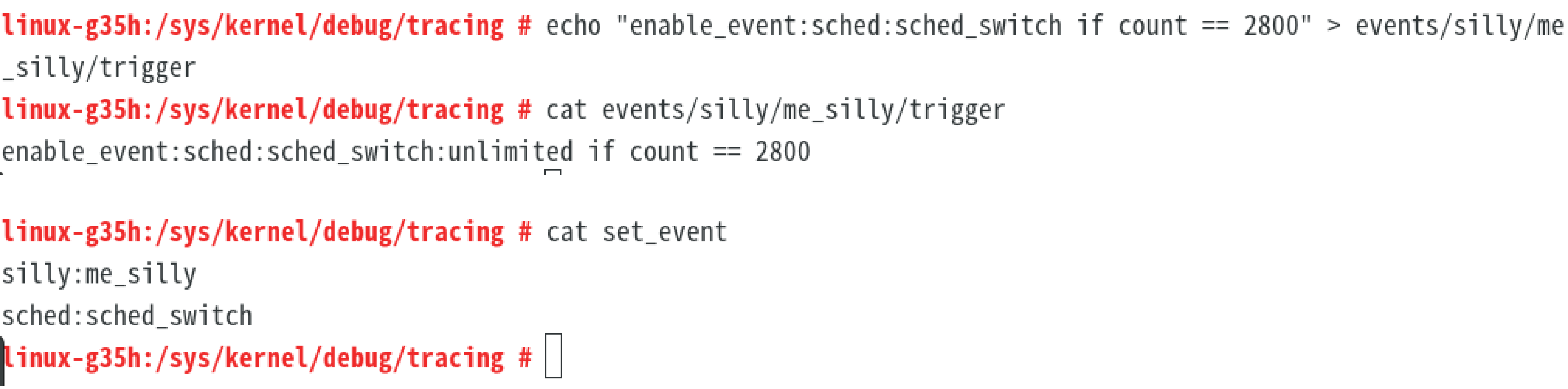
```
linux-g35h:/sys/kernel/debug/tracing/events/silly/me_silly # cat format
name: me_silly
ID: 1045
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3;       size:1; signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:unsigned long time;       offset:8;       size:8; signed:0;
        field:unsigned long count;      offset:16;      size:8; signed:0;

print fmt: "time=%lu count=%lu", REC->time, REC->count
linux-g35h:/sys/kernel/debug/tracing/events/silly/me_silly # echo "count > 500 && count <= 510" > filter
linux-g35h:/sys/kernel/debug/tracing/events/silly/me_silly # cat filter
count > 500 && count <= 510
linux-g35h:/sys/kernel/debug/tracing/events/silly/me_silly # echo 0 > filter
linux-g35h:/sys/kernel/debug/tracing/events/silly/me_silly # cat filter
none
```

Event filter result



```
linux-g35h:~ # cat /sys/kernel/debug/tracing/trace_pipe
    silly-thread-1829  [000] ...1  1464.155686: me_silly: time=4295258308 count=501
    silly-thread-1829  [000] ...1  1465.155654: me_silly: time=4295258558 count=502
    silly-thread-1829  [000] ...1  1466.155624: me_silly: time=4295258808 count=503
    silly-thread-1829  [000] ...1  1467.155585: me_silly: time=4295259058 count=504
    silly-thread-1829  [000] ...1  1468.155536: me_silly: time=4295259308 count=505
    silly-thread-1829  [000] ...1  1469.155488: me_silly: time=4295259558 count=506
    silly-thread-1829  [000] ...1  1470.155402: me_silly: time=4295259808 count=507
    silly-thread-1829  [000] ...1  1471.155361: me_silly: time=4295260058 count=508
    silly-thread-1829  [000] ...1  1472.155382: me_silly: time=4295260308 count=509
    silly-thread-1829  [000] ...1  1473.155343: me_silly: time=4295260558 count=510
```

Setting event trigger



```
linux-g35h:/sys/kernel/debug/tracing # echo "enable_event:sched:sched_switch if count == 2800" > events/silly/me
_silly/trigger
linux-g35h:/sys/kernel/debug/tracing # cat events/silly/me_silly/trigger
enable_event:sched:sched_switch:unlimited if count == 2800

linux-g35h:/sys/kernel/debug/tracing # cat set_event
silly:me_silly
sched:sched_switch
linux-g35h:/sys/kernel/debug/tracing #
```

Q&A

Reference

- [1] Documentation/trace/tracepoints.txt, Mathieu Desnoyers, Linux Kernel
- [2] Documentation/trace/tracepoint-analysis.txt, Mel Gorman, Linux Kernel
- [3] Using the TRACE_EVENT() macro (Part 1)
- [4] Using the TRACE_EVENT() macro (Part 2)
- [5] Using the TRACE_EVENT() macro (Part 3)

  Steven Rostedt, LWN.net, March, 2010
- [6] Documentation/trace/events.txt, Theodore Ts'o, Linux Kernel

Feedback to
jlee@suse.com

Thank you.

We adapt. You succeed.

We adapt. You succeed.

We adapt. You succeed.

Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org