Using the TRACE_EVENT() macro

August, 2017, Beijing

Joey Lee
SUSE Labs Taipei

Agenda

- Steven Rostedt's "Using the TRACE_EVENT() macro" articles
- sched_switch TRACE_EVENT review
- TRACE_EVENT() tricks
- sillymod kernel module
- Q&A

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/

sched_switch TRACE_EVENT review

sched_switch trace event

- include/trace/events/sched.h

```
/*
 * Tracepoint for task switches, performed by the scheduler:
 */
TRACE_EVENT(sched_switch,

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

        TP_ARGS(preempt, prev, next),

        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                       )
        ),

        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 */
        ),

        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" },
```

V4.12 mainline kernel

sched_switch trace event (name)

- name - the name of the tracepoint to be created. [1]

```
/*
 * 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. [1]

```
#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. [1]
- This parameter describes the structure layout of the data that will be stored in the tracer's ring buffer. [1]

```
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. [1]

```
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. [1]
- The TRACE_HEADER_MULTI_READ test allows this file to be included more than once. [1]
- 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. [1]

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. [1]

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! [2]
- 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. [2]

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));
```

TRACE_EVENT() tricks

cpp tricks

```
#define DOGS { C(JACK_RUSSELL), C(BULL_TERRIER),
C(ITALIAN_GREYHOUND) }
#undef C
#define C(a) ENUM_##a
enum dog_enums DOGS;
#undef C
#define C(a) #a
char *dog_strings[] = DOGS;
char *dog_to_string(enum dog_enums dog)
{
        return dog_strings[dog];
}
```

- The trick is that the macro DOGS contains a sub-macro C() that we can redefine and change the behavior of DOGS. This concept is key to how the TRACE_EVENT() macro works. [3]

Redefine TRACE_EVENT()

- All the sub-macros within TRACE_EVENT() can be redefined and cause the TRACE_EVENT() to be used to create different code that uses the same information.
- The tracepoint code created by Mathieu Desnoyers required a DECLARE_TRACE(name, proto, args) be defined in a header file, and in some C file DEFINE_TRACE(name) was used. TRACE_EVENT() now does both jobs. [3]

kernel/sched/core.c

Set to redefine TRACE_EVENT by DEFINE_TRACE to overwrite tracepoint.h

```
#define CREATE_TRACE_POINTS
#include <trace/events/sched.h>
[...snip]
static void __sched notrace __schedule(bool preempt)
{
        struct task_struct *prev, *next;
        unsigned long *switch_count;
[...snip]
        if (likely(prev != next)) {
                rq->nr_switches++;
                rq->curr = next;
                ++*switch_count;

                trace_sched_switch(preempt, prev, next);
[...snip]
```

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>
#include <linux/binfmts.h>
[...snip]
/*
 * Tracepoint for task switches, performed by the scheduler:
 */
TRACE_EVENT(sched_switch,

        TP_PROTO(bool preempt,
[...snip]
#endif /* _TRACE_SCHED_H */

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

include/linux/tracepoint.h

```
#ifndef _LINUX_TRACEPOINT_H
#define _LINUX_TRACEPOINT_H
[...snip]
#endif /* _LINUX_TRACEPOINT_H */

#ifndef DECLARE_TRACE
[...snip]
#define __DECLARE_TRACE(name, proto, args, cond, data_proto, data_args) \
        extern struct tracepoint __tracepoint_##name;                   \
        static inline void trace_##name(proto)                          \
        {                                                               \
//[...snip]
#define DEFINE_TRACE_FN(name, reg, unreg)                                \
//[...snip]
        struct tracepoint __tracepoint_##name                            \
        __attribute__((section("__tracepoints"))) =                      \
                { __tpstrtab_##name, STATIC_KEY_INIT_FALSE, reg, unreg, NULL };\
//[...snip]
#define DEFINE_TRACE(name)                                              \
        DEFINE_TRACE_FN(name, NULL, NULL);
//[..snip]
#define DECLARE_TRACE(name, proto, args)                                \
        __DECLARE_TRACE(name, PARAMS(proto), PARAMS(args),              \
//[...snip]
#define TRACE_EVENT(name, proto, args, struct, assign, print)   \
        DECLARE_TRACE(name, PARAMS(proto), PARAMS(args))
#endif /* ifdef TRACE_EVENT (see note above) */
```

First defined TRACE_EVENT by DECLARE_TRACE

include/trace/define_trace.h

```
/*
 * Trace files that want to automate creation of all tracepoints defined
 * in their file should include this file. The following are macros that the
 * trace file may define:
 *
 * TRACE_SYSTEM defines the system the tracepoint is for
 *
 * TRACE_INCLUDE_FILE if the file name is something other than TRACE_SYSTEM.h
 *     This macro may be defined to tell define_trace.h what file to include.
 *     Note, leave off the ".h".
 *
 * TRACE_INCLUDE_PATH if the path is something other than core kernel include/trace
 *     then this macro can define the path to use. Note, the path is relative to
 *     define_trace.h, not the file including it. Full path names for out of tree
 *     modules must be used.
 */

#ifdef CREATE_TRACE_POINTS

/* Prevent recursion */
#undef CREATE_TRACE_POINTS
[...snip]
#undef TRACE_EVENT
#define TRACE_EVENT(name, proto, args, tstruct, assign, print)  \
        DEFINE_TRACE(name)
[...snip]
#undef TRACE_INCLUDE
#undef __TRACE_INCLUDE

#ifndef TRACE_INCLUDE_FILE
# define TRACE_INCLUDE_FILE TRACE_SYSTEM
# define UNDEF_TRACE_INCLUDE_FILE
#endif
```

Set in kernel/sched/core.c before Including include/trace/events/sched.h

Redefine TRACE_EVENT by DEFINE_TRACE to overwrite tracepoint.h

include/trace/define_trace.h (cont.)

```
# define __TRACE_INCLUDE(system) <trace/events/system.h>
# define UNDEF_TRACE_INCLUDE_PATH
#else
# define __TRACE_INCLUDE(system) __stringify(TRACE_INCLUDE_PATH/system.h)
#endif

# define TRACE_INCLUDE(system) __TRACE_INCLUDE(system)

/* Let the trace headers be reread */
#define TRACE_HEADER_MULTI_READ

#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)
[...snip]
/* Only undef what we defined in this file */
#ifdef UNDEF_TRACE_INCLUDE_FILE
# undef TRACE_INCLUDE_FILE
# undef UNDEF_TRACE_INCLUDE_FILE
#endif

#ifdef UNDEF_TRACE_INCLUDE_PATH
# undef TRACE_INCLUDE_PATH
# undef UNDEF_TRACE_INCLUDE_PATH
#endif

/* We may be processing more files */
#define CREATE_TRACE_POINTS

#endif /* CREATE_TRACE_POINTS */
```

#include <trace/events/sched.h> again

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>
#include <linux/binfmts.h>
[...snip]
/*
 * Tracepoint for task switches, performed by the scheduler:
 */
TRACE_EVENT(sched_switch,

        TP_PROTO(bool preempt,
[...snip]
#endif /* _TRACE_SCHED_H */

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

Allow #include <trace/events/sched.h> again

Skipped when re-entering

Redefine TRACE_EVENT by DEFINE_TRACE because define_trace.h

include/linux/tracepoint.h again

```
#ifndef _LINUX_TRACEPOINT_H
#define _LINUX_TRACEPOINT_H
[...snip]
#endif /* _LINUX_TRACEPOINT_H */

/*
 * Note: we keep the TRACE_EVENT and DECLARE_TRACE outside the include
 *  file ifdef protection.
 *  This is due to the way trace events work. If a file includes two
 *  trace event headers under one "CREATE_TRACE_POINTS" the first include
 *  will override the TRACE_EVENT and break the second include.
 */

#ifndef DECLARE_TRACE
[...snip]
#define __DECLARE_TRACE(name, proto, args, cond, data_proto, data_args) \
        extern struct tracepoint __tracepoint_##name;                   \
[...snip]
#define DECLARE_TRACE(name, proto, args)                                \
        __DECLARE_TRACE(name, PARAMS(proto), PARAMS(args),              \
                        cpu_online(raw_smp_processor_id()),             \
                        PARAMS(void *__data, proto),                    \
                        PARAMS(__data, args))
[...snip]
#endif /* DECLARE_TRACE */

#ifndef TRACE_EVENT
[...snip]
#define TRACE_EVENT(name, proto, args, struct, assign, print)   \
        DECLARE_TRACE(name, PARAMS(proto), PARAMS(args))
#endif /* ifdef TRACE_EVENT (see note above) */
```

Skipped when re-entering

include/trace/define_trace.h

```
# define TRACE_INCLUDE(system) __TRACE_INCLUDE(system)

/* Let the trace headers be reread */
#define TRACE_HEADER_MULTI_READ

#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)

/* Make all open coded DECLARE_TRACE nops */
#undef DECLARE_TRACE
#define DECLARE_TRACE(name, proto, args)

#ifdef TRACEPOINTS_ENABLED
#include <trace/trace_events.h>
#include <trace/perf.h>
#endif

#undef TRACE_EVENT
#undef TRACE_EVENT_FN
#undef TRACE_EVENT_FN_COND
```

Overwrite TRACE_EVENT definition again

include/trace/trace_events.h

```
/*
 * Stage 1 of the trace events.
 *
 * Override the macros in <trace/trace_events.h> to include the following:
 *
 * struct trace_event_raw_<call> {
[...snip]
 */

#include <linux/trace_events.h>

#ifndef TRACE_SYSTEM_VAR
#define TRACE_SYSTEM_VAR TRACE_SYSTEM
#endif
[...snip]
#undef TRACE_DEFINE_ENUM
#define TRACE_DEFINE_ENUM(a)                            \
[...snip]
#undef TRACE_EVENT
#define TRACE_EVENT(name, proto, args, tstruct, assign, print) \
        DECLARE_EVENT_CLASS(name,                               \
[...snip]
#undef DECLARE_EVENT_CLASS
#define DECLARE_EVENT_CLASS(name, proto, args, tstruct, assign, print)  \
        struct trace_event_raw_##name {                                 \
[...snip]
#undef DEFINE_EVENT
#define DEFINE_EVENT(template, name, proto, args)       \
        static struct trace_event_call  __used          \
        __attribute__((__aligned__(4))) event_##name
[...snip]
#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)

/*
 * Stage 2 of the trace events.
 *
 * Include the following:
```

#include <trace/events/sched.h> again

include/trace/trace_events.h (cont.)

```
#undef DEFINE_EVENT
#define DEFINE_EVENT(template, name, proto, args)

#undef DEFINE_EVENT_PRINT
#define DEFINE_EVENT_PRINT(template, name, proto, args, print)  \
        DEFINE_EVENT(template, name, PARAMS(proto), PARAMS(args))

#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)

/*
 * Stage 4 of the trace events.
 *
 * Override the macros in <trace/trace_events.h> to include the following:
[...snip]
 */

#ifdef CONFIG_PERF_EVENTS
[...snip]
#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)
[...snip]
static struct trace_event_call __used                                   \
__attribute__((section("_ftrace_events"))) *__event_##call = &event_##call

#include TRACE_INCLUDE(TRACE_INCLUDE_FILE)
```

#include <trace/events/sched.h> again

#include <trace/events/sched.h> again

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

Steven Rostedt [3]

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

```
#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();
```

Q&A

Reference

- [1] Using the TRACE_EVENT() macro (Part 1)
- [2] Using the TRACE_EVENT() macro (Part 2)
- [3] Using the TRACE_EVENT() macro (Part 3)

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

Feedback to
jlee@suse.com

Thank you.


SUSE®
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