The Linux Kernel Tracepoint API

Jason Baron

jbaron@redhat.com

The Linux Kernel Tracepoint API

by Jason Baron

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

Table of Contents

Chapter 1. Introduction

Tracepoints are static probe points that are located in strategic points throughout the kernel. ’Probes’ register/unregister with tracepoints via a callback mechanism. The ’probes’ are strictly typed functions that are passed a unique set of parameters defined by each tracepoint.

From this simple callback mechanism, ’probes’ can be used to profile, debug, and understand kernel behavior. There are a number of tools that provide a framework for using ’probes’. These tools include Systemtap, ftrace, and LTTng.

Tracepoints are defined in a number of header files via various macros. Thus, the purpose of this document is to provide a clear accounting of the available tracepoints. The intention is to understand not only what tracepoints are available but also to understand where future tracepoints might be added.

The API presented has functions of the form: `trace_tracepointname(function parameters)`. These are the tracepoints callbacks that are found throughout the code. Registering and unregistering probes with these callback sites is covered in the `Documentation/trace/*` directory.

Chapter 2. IRQ

trace_irq_handler_entry

LINUX

Kernel Hackers ManualMay 2010

Name

`trace_irq_handler_entry` — called immediately before the irq action handler

Synopsis

```
void trace_irq_handler_entry (int irq, struct irqaction *
action);
```

Arguments

irq

irq number

action

pointer to struct irqaction

Description

The struct irqaction pointed to by *action* contains various information about the handler, including the device name, *action*->name, and the device id, *action*->dev_id. When used in conjunction with the irq_handler_exit tracepoint, we can figure out irq handler latencies.

trace_irq_handler_exit

LINUX

Kernel Hackers ManualMay 2010

Name

`trace_irq_handler_exit` — called immediately after the irq action handler returns

Synopsis

```
void trace_irq_handler_exit (int irq, struct irqaction *
action, int ret);
```

Arguments

irq

irq number

action

pointer to struct irqaction

ret

return value

Description

If the *ret* value is set to IRQ_HANDLED, then we know that the corresponding *action*->handler scuccessully handled this irq. Otherwise, the irq might be a shared irq line, or the irq was not handled successfully. Can be used in conjunction with the irq_handler_entry to understand irq handler latencies.

trace_softirq_entry

LINUX

Kernel Hackers ManualMay 2010

Name

`trace_softirq_entry` — called immediately before the softirq handler

Synopsis

```
void trace_softirq_entry (struct softirq_action * h, struct
softirq_action * vec);
```

Arguments

h

pointer to struct softirq_action

vec

pointer to first struct softirq_action in softirq_vec array

Description

The *h* parameter, contains a pointer to the struct softirq_action which has a pointer to the action handler that is called. By subtracting the *vec* pointer from the *h* pointer, we can determine the softirq number. Also, when used in combination with the softirq_exit tracepoint we can determine the softirq latency.

trace_softirq_exit

LINUX

Kernel Hackers ManualMay 2010

Name

`trace_softirq_exit` — called immediately after the softirq handler returns

Synopsis

```
void trace_softirq_exit (struct softirq_action * h, struct
softirq_action * vec);
```

Arguments

h

pointer to struct softirq_action

vec

pointer to first struct softirq_action in softirq_vec array

Description

The *h* parameter contains a pointer to the struct softirq_action that has handled the softirq. By subtracting the *vec* pointer from the *h* pointer, we can determine the softirq number. Also, when used in combination with the softirq_entry tracepoint we can determine the softirq latency.