

🏠 The Linux Kernel	
5.0.0	
The Linux kernel user’s and administrator’s guide	
The Linux kernel user-space API guide	
Working with the kernel development community	
Development tools for the kernel	
How to write kernel documentation	
Kernel Hacking Guides	
☰ Linux Tracing Technologies	
Function Tracer Design	
Notes on Analysing Behaviour Using Events and Tracepoints	
ftrace - Function Tracer	
Using ftrace to hook to functions	
☰ Kprobe-based Event Tracing	
Overview	
Synopsis of kprobe_events	
Types	
Per-Probe Event Filtering	
Event Profiling	
Usage examples	
Uprobe-tracer: Uprobe-based Event Tracing	
Using the Linux Kernel Tracepoints	
Event Tracing	
Subsystem Trace Points: kmem	
Subsystem Trace Points: power	
NMI Trace Events	
MSR Trace Events	
In-kernel memory-mapped I/O tracing	
Event Histograms	
Hardware Latency Detector	
Intel(R) Trace Hub (TH)	
System Trace Module	
MIPI SyS-T over STP	
Kernel Maintainer Handbook	
The Linux driver implementer’s API guide	
Core API Documentation	
Linux Media Subsystem Documentation	
Linux Networking Documentation	
The Linux Input Documentation	
Linux GPU Driver Developer’s Guide	
Security Documentation	
Linux Sound Subsystem Documentation	
Linux Kernel Crypto API	

type> (e.g. +0(%di):x32[1] is same as +0(%di):x32.) But string[1] is not equal to string. The string type itself represents “char array”, but string array type represents “char * array”. So, for example, +0(%di):string[1] is equal to +0(+0(%di)):string. Bitfield is another special type, which takes 3 parameters, bit-width, bit- offset, and container-size (usually 32). The syntax is:

```
b<bit-width>@<bit-offset>/<container-size>
```

Symbol type(‘symbol’) is an alias of u32 or u64 type (depends on BITS_PER_LONG) which shows given pointer in “symbol+offset” style. For \$comm, the default type is “string”; any other type is invalid.

Per-Probe Event Filtering

Per-probe event filtering feature allows you to set different filter on each probe and gives you what arguments will be shown in trace buffer. If an event name is specified right after ‘p:’ or ‘r:’ in kprobe_events, it adds an event under tracing/events/kprobes/<EVENT>, at the directory you can see ‘id’, ‘enable’, ‘format’, ‘filter’ and ‘trigger’.

enable:
You can enable/disable the probe by writing 1 or 0 on it.

format:
This shows the format of this probe event.

filter:
You can write filtering rules of this event.

id:
This shows the id of this probe event.

trigger:
This allows to install trigger commands which are executed when the event is hit (for details, see Documentation/trace/events.rst, section 6).

Event Profiling

You can check the total number of probe hits and probe miss-hits via /sys/kernel/debug/tracing/kprobe_profile. The first column is event name, the second is the number of probe hits, the third is the number of probe miss-hits.

Usage examples

To add a probe as a new event, write a new definition to kprobe_events as below:

```
echo 'p:myprobe do_sys_open dfd=%ax filename=%dx flags=%cx mode=+4($stack)' > /sys/kernel/debug/tracing/kprobe_events
```


This sets a kprobe on the top of do_sys_open() function with recording 1st to 4th arguments as “myprobe” event. Note, which register/stack entry is assigned to each function argument depends on arch-specific ABI. If you unsure the ABI, please try to use probe subcommand of perf-tools (you can find it under tools/perf). As this example shows, users can choose more familiar names for each arguments.

```
echo 'r:myretprobe do_sys_open $retval' >> /sys/kernel/debug/tracing/kprobe_events
```

This sets a kretprobe on the return point of do_sys_open() function with recording return value as “myretprobe” event. You can see the format of these events via /sys/kernel/debug/tracing/events/kprobes/<EVENT>/format.

```
cat /sys/kernel/debug/tracing/events/kprobes/myprobe/format
name: myprobe
ID: 780
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 __probe_ip; offset:12;      size:4; signed:0;
    field:int __probe_nargs;  offset:16;      size:4; signed:1;
    field:unsigned long dfd;  offset:20;      size:4; signed:0;
```

The Linux Kernel

5.0.0

The Linux kernel user’s and administrator’s guide

The Linux kernel user-space API guide

Working with the kernel development community

Development tools for the kernel

How to write kernel documentation

Kernel Hacking Guides

Linux Tracing Technologies

Function Tracer Design

Notes on Analysing Behaviour Using Events and Tracepoints

ftrace - Function Tracer

Using ftrace to hook to functions

Kprobe-based Event Tracing

Overview

Synopsis of kprobe_events

Types

Per-Probe Event Filtering

Event Profiling

Usage examples

Uprobe-tracer: Uprobe-based Event Tracing

Using the Linux Kernel Tracepoints

Event Tracing

Subsystem Trace Points: kmem

Subsystem Trace Points: power

NMI Trace Events

MSR Trace Events

In-kernel memory-mapped I/O tracing

Event Histograms

Hardware Latency Detector

Intel(R) Trace Hub (TH)

System Trace Module

MIPI SyS-T over STP

Kernel Maintainer Handbook

The Linux driver implementer’s API guide

Core API Documentation

Linux Media Subsystem Documentation

Linux Networking Documentation

The Linux Input Documentation

Linux GPU Driver Developer’s Guide

Security Documentation

Linux Sound Subsystem Documentation

Linux Kernel Crypto API

```
field:unsigned long filename;    offset:24;    size:4; signed:0;
field:unsigned long flags;      offset:28;    size:4; signed:0;
field:unsigned long mode;       offset:32;    size:4; signed:0;

print fmt: "(%lx) dfd=%lx filename=%lx flags=%lx mode=%lx", REC->__probe_ip,
REC->dfd, REC->filename, REC->flags, REC->mode
```

You can see that the event has 4 arguments as in the expressions you specified.

```
echo > /sys/kernel/debug/tracing/kprobe_events
```

This clears all probe points.

Or,

```
echo -:myprobe >> kprobe_events
```

This clears probe points selectively.

Right after definition, each event is disabled by default. For tracing these events, you need to enable it.

```
echo 1 > /sys/kernel/debug/tracing/events/kprobes/myprobe/enable
echo 1 > /sys/kernel/debug/tracing/events/kprobes/myretprobe/enable
```

And you can see the traced information via /sys/kernel/debug/tracing/trace.

```
cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
#          TASK-PID    CPU#    TIMESTAMP    FUNCTION
#          | |        |         |          |
<...>-1447 [001] 1038282.286875: myprobe: (do_sys_open+0x0/0xd6) dfd=3 filename:
<...>-1447 [001] 1038282.286878: myretprobe: (sys_openat+0xc/0xe <- do_sys_open
<...>-1447 [001] 1038282.286885: myprobe: (do_sys_open+0x0/0xd6) dfd=ffffff9c f
<...>-1447 [001] 1038282.286915: myretprobe: (sys_open+0x1b/0x1d <- do_sys_open
<...>-1447 [001] 1038282.286969: myprobe: (do_sys_open+0x0/0xd6) dfd=ffffff9c f
<...>-1447 [001] 1038282.286976: myretprobe: (sys_open+0x1b/0x1d <- do_sys_open
```

Each line shows when the kernel hits an event, and <- SYMBOL means kernel returns from SYMBOL(e.g. “sys_open+0x1b/0x1d <- do_sys_open” means kernel returns from do_sys_open to sys_open+0x1b).

Previous

Next

© Copyright The kernel development community.

Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

Page 3 of 3