

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

[Docs](#) » [Linux Tracing Technologies](#) » Kprobe-based Event Tracing

[View page source](#)

Kprobe-based Event Tracing

Author: Masami Hiramatsu

Overview

These events are similar to tracepoint based events. Instead of Tracepoint, this is based on kprobes (kprobe and kretprobe). So it can probe wherever kprobes can probe (this means, all functions except those with `__kprobes/nokprobe_inline` annotation and those marked `NOKPROBE_SYMBOL`). Unlike the Tracepoint based event, this can be added and removed dynamically, on the fly.

To enable this feature, build your kernel with `CONFIG_KPROBE_EVENTS=y`.


Similar to the events tracer, this doesn't need to be activated via `current_tracer`. Instead of that, add probe points via `/sys/kernel/debug/tracing/kprobe_events`, and enable it via `/sys/kernel/debug/tracing/events/kprobes/<EVENT>/enable`.

You can also use `/sys/kernel/debug/tracing/dynamic_events` instead of `kprobe_events`. That interface will provide unified access to other dynamic events too.

Synopsis of kprobe_events

```
p[:[GRP/]EVENT] [MOD:]SYM[+offs]|MEMADDR [FETCHARGS] :  
r[MAXACTIVE][:[GRP/]EVENT] [MOD:]SYM[+0] [FETCHARGS] :  
-:[GRP/]EVENT :
```

GRP : Group name. If omitted, use "kprobes" for
EVENT : Event name. If omitted, the event name is based on SYM+offs or MEMADDR.
MOD : Module name which has given SYM.

 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

SYM[+offs]	: Symbol+offset where the probe is inserted.
MEMADDR	: Address where the probe is inserted.
MAXACTIVE	: Maximum number of instances of the specification can be probed simultaneously, or 0 for unlimited as defined in Documentation/kprobes.txt
FETCHARGS	: Arguments. Each probe can have up to 128 arguments.
%REG	: Fetch register REG
@ADDR	: Fetch memory at ADDR (ADDR should be in kernel space)
@SYM[+ -offs]	: Fetch memory at SYM + - offs (SYM should be in kernel space)
\$stackN	: Fetch Nth entry of stack (N >= 0)
\$stack	: Fetch stack address.
\$argN	: Fetch the Nth function argument. (N >= 1)
\$retval	: Fetch return value.(*2)
\$comm	: Fetch current task comm.
+ -offs(FETCHARG)	: Fetch memory at FETCHARG + - offs address
NAME=FETCHARG	: Set NAME as the argument name of FETCHARG
FETCHARG:TYPE	: Set TYPE as the type of FETCHARG. Current types are: (u8/u16/u32/u64/s8/s16/s32/s64), hexadecimal (x8/x16/x32/x64), "string" and bitfield
(*1)	only for the probe on function entry (offs == 0).
(*2)	only for return probe.
(*3)	this is useful for fetching a field of data structure

Types

Several types are supported for fetch-args. Kprobe tracer will access memory by given type. Prefix 's' and 'u' means those types are signed and unsigned respectively. 'x' prefix implies it is unsigned. Traced arguments are shown in decimal ('s' and 'u') or hexadecimal ('x'). Without type casting, 'x32' or 'x64' is used depends on the architecture (e.g. x86-32 uses x32, and x86-64 uses x64). These value types can be an array. To record array data, you can add '[N]' (where N is a fixed number, less than 64) to the base type. E.g. 'x16[4]' means an array of x16 (2bytes hex) with 4 elements. Note that the array can be applied to memory type fetchargs, you can not apply it to registers/stack-entries etc. (for example, '\$stack1:x8[8]' is wrong, but '+8(\$stack):x8[8]' is OK.) String type is a special type, which fetches a "null-terminated" string from kernel space. This means it will fail and store NULL if the string container has been paged out. The string array type is a bit different from other types. For other base types, <base-type>[1] is equal to <base-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

Page 2 of 6

🏠 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

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.

🏠 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

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=%c
```

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/de
```

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/form
name: myprobe
ID: 780
format:
    field:unsigned short common_type;      offset:0;
    field:unsigned char common_flags;      offset:2;
    field:unsigned char common_preempt_count;  o
    field:int common_pid;   offset:4;      size:4; s

    field:unsigned long __probe_ip; offset:12;    s
    field:int __probe_nargs;      offset:16;      s
    field:unsigned long dfd;      offset:20;      s
    field:unsigned long filename; offset:24;      s
    field:unsigned long flags;    offset:28;      s
    field:unsigned long mode;     offset:32;      s

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

🏠 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

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

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
<...>-1447 [001] 1038282.286878: myretprobe:
<...>-1447 [001] 1038282.286885: myprobe: (do
<...>-1447 [001] 1038282.286915: myretprobe:
<...>-1447 [001] 1038282.286969: myprobe: (do
<...>-1447 [001] 1038282.286976: myretprobe:
```

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 ➡

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

© Copyright The kernel development community.

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

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