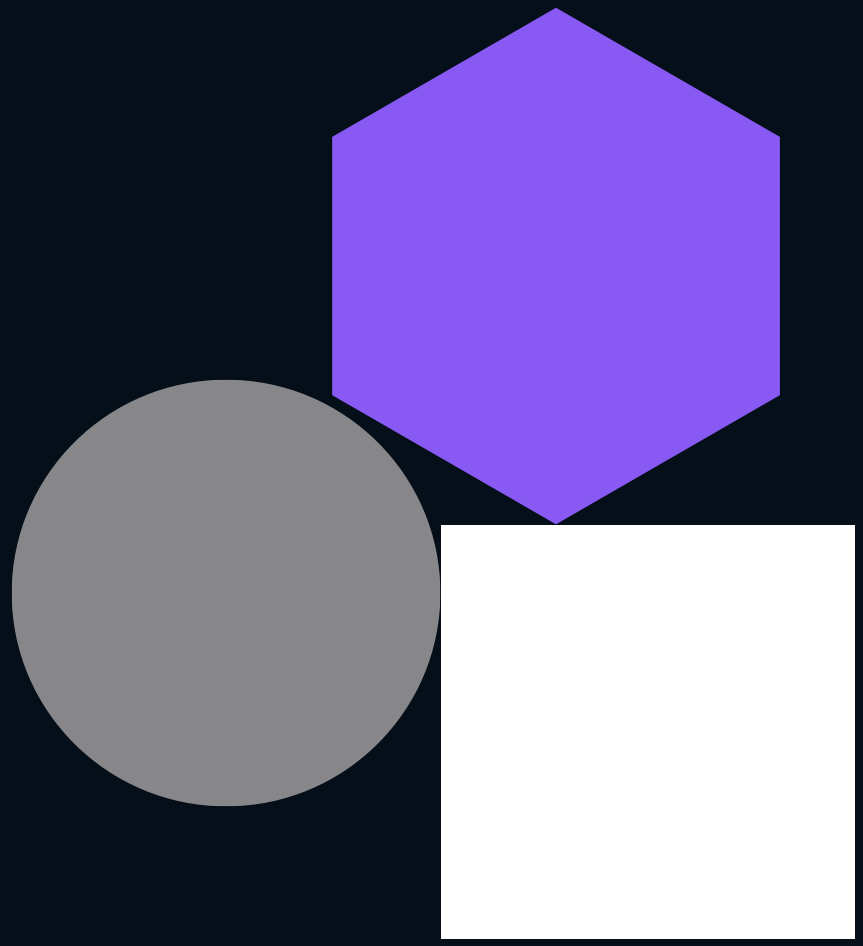
ftrace on cheri linux

Codasip



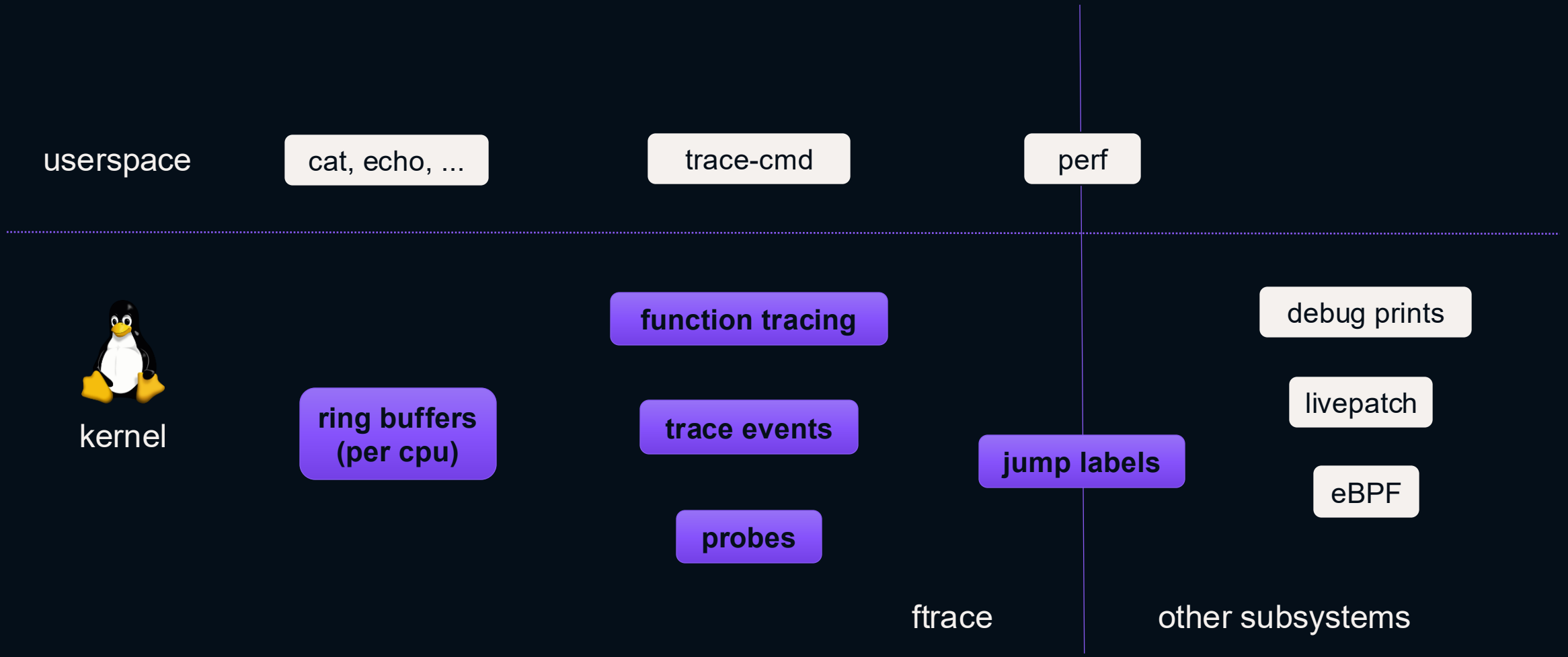
porting a fairly complex kernel subsystem

Martin Kaiser

CHERITech'26, Munich

→ ftrace in the linux kernel

Codasip



© 2026 Codasip. All rights reserved.

→ ring buffer

Codasip

- no capabilities in the ring buffer
  - increased alignment wastes too much space
  - don't copy kernel capabilities to userspace

© 2026 Codasip. All rights reserved.

→ function tracing

- "function X calls function Y"
  - dynamic ftrace: enable / disable tracing for each target function
- *-fpatchable-function-entry*
  - insert nop instructions
- trampoline reads ftrace_ops
  - access ftrace_ops relative to ct0
  - but ct0 is a sealed capability
- dirty hack: fabricate new capability
  - pcc's metadata, ct0's address
  - must be revisited when pcc bounds are tightened

```
root@codasip-prime:/sys/kernel/tracing# cat trace
# tracer: function
#
...
  sh-137   [000] ... 25.029198: vfs_write <- ksys_write
```

```
ffffffff803ba3d0: 0001   nop
ffffffff803ba3d2: 0001   nop
...
ffffffff803ba3dc: 0001   nop
ffffffff803ba3de: 0001   nop
ffffffff803ba3e0 <vfs_write>:
ffffffff803ba3e0: 0001   nop
ffffffff803ba3e2: 0001   nop
ffffffff803ba3e4: 0001   nop
ffffffff803ba3e6: 0001   nop      jalr ct0, ...
ffffffff803ba3e8: 714d   caddi    csp, csp, -0x150
ffffffff803ba3ea: a286   sc       cra, 0x140(csp)
...
```

struct ftrace_ops

jump to trampoline

```
… read ftrace_ops
… write log entry
jr ct0
```

© 2026 Codasip. All rights reserved.

→ function graph tracing

Codasip

- keep track of nested function calls
- hook into start and end of a traced function
  - ftrace start code replaces the saved return address
  - ftrace end code restores it
- ret_stack
  - saved return addresses
  - other data to be passed from start to end hooks
- for cheri, ret_stack must be a capability array

```
root@codasip-prime:/sys/kernel/tracing#
cat trace
# tracer: function_graph
...
1)               |  vfs_write() {
1)               |    tty_write() {
1) # 214.023 us  |      file_tty_write();
1) # 271.386 us  |    }
1) # 337.700 us  |  }
```

© 2026 Codasip. All rights reserved.

→ events

Codasip

- predefined "points of interest"
  - collect data
  - write a log entry
- event definition
  - event's input data
  - what is written into the ring buffer
  - format of a parsed ring buffer entry
- lots of events are logging pointers
  - two numeric fields: address, tag+metadata
- parse & display a ring buffer entry
  - combine address and tag+meta, pass them to printk

```
TRACE_EVENT(hrtimer_expire_entry,

    TP_PROTO(struct hrtimer *hrtimer, ktime_t now),

    TP_ARGS(hrtimer, now),

    TP_STRUCT__entry(
        __field( void *,    hrtimer )
        __field( s64,       now     )
        ...
    ),

    TP_fast_assign(
        __entry->hrtimer    = hrtimer;
        __entry->now        = now;
        ...
    ),

    TP_printk("hrtimer=%p ... now=%llu",
          __entry->hrtimer, …, (unsigned long long) __entry->now)
);
```

© 2026 Codasip. All rights reserved.

→ events

Codasip

- event definitions must be patched
  - conversion script
  - event injection
    - generate arbitrary events for testing

```
TRACE_EVENT(hrtimer_expire_entry,

    TP_PROTO(struct hrtimer *hrtimer, ktime_t now),

    TP_ARGS(hrtimer, now),

    TP_STRUCT__entry(
          __ptr( void *,    hrtimer )
        __field( s64,       now     )
        ...
    ),

    TP_fast_assign(
        __assign_ptr(hrtimer, hrtimer);
        __entry->now        = now;
        ...
    ),

    TP_printk("hrtimer= " TRACE_CAP_FMT ... "now=%llu",
          __get_cap(hrtimer), ..., (unsigned long long) __entry->now)
);
```

© 2026 Codasip. All rights reserved.

→ jump labels

- event processing
- checking must be efficient
  - lots of events, mostly disabled
  - ok if changing the state (enabled / disabled) is slow
- generate code for the initial setting
  - patch the code if the state is changed
- data types
  - static_key : an enabled / disabled switch
  - jump_entry : a caller

```
if (event_is_enabled)
   … write a log entry
```



```
   nop
X: …
   …

Y: write a log entry
   jump back to X
```

© 2026 Codasip. All rights reserved.

→ jump labels

Codasip

- a static_key links to a list of callers
  - LSB encodes a "type"
- update the list of callers

```
type = key->type & 0x1;
key->entries = entries;
key->type |= type;
```

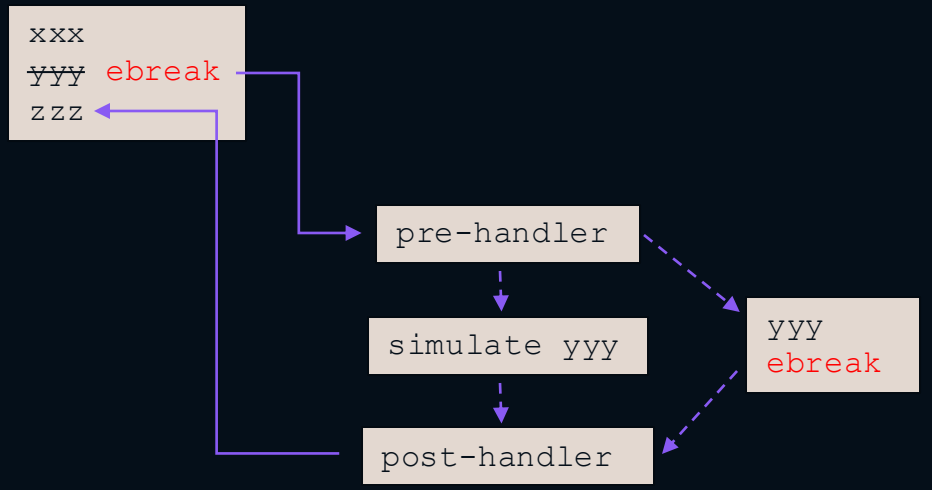
→ this clears the tag of key->entries

```
struct static_key {
    atomic_t enabled;
    union {
        unsigned long type;
        struct jump_entry *entries;
        ...
    };
};
```

© 2026 Codasip. All rights reserved.

→ probes

- create a dynamic event
  - callback function collects data for the event
- fprobe, kprobe, uprobe, …
- kprobe
  - can be placed at (almost) any assembly instruction in the kernel
  - replace the instruction with ebreak
  - execute it at a different location
  - if the instruction depends on PC: simulate it in C code



© 2026 Codasip. All rights reserved.

→ probes

Codasip

- cheri
  - new cheri instructions need no simulation
  - update the simulation functions
    - e.g. create sealed capability for jalr's return address
- configuration of the dynamic event
  - what data should be collected (memory, registers, …)?
    - new fetch type: capability

```
$ echo 'p n_tty_write+0x32 reg_a0=%a0:cap' > /sys/kernel/tracing/dynamic_events
$ trace-cmd start -e kprobes ; echo test123 ; trace-cmd stop ; trace-cmd show
...
  sh-189 [000] ... 14653.127340: p_n_tty_write_50: (n_tty_write+0x32/0x460)
                                 reg_a0=0xffffffd683ecd000[V:0000:C:rw.C.lse:1:.:0xffffffd683ecd000-0xffffffd683ece000]]
```

© 2026 Codasip. All rights reserved.

→ summary

Codasip

- dynamic function tracing
- cheri support for trace events
- jump labels for events and dynamic debug prints
- f/k/u/t/eprobes are working
  - capability fetch type
- linux kernel for cheri: https://github.com/CHERI-Alliance/linux
- userspace utilities: trace-cmd, perf
- please let us know what's missing or broken

© 2026 Codasip. All rights reserved.



Thank you!

https://github.com/CHERI-Alliance/

martin.kaiser@codasip.com

© 2026 Codasip. All rights reserved.