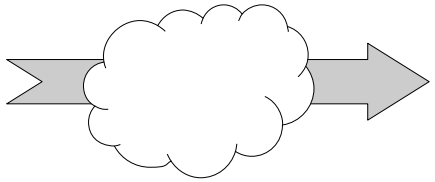
# where have all the kprobes gone

---

jiri olsa / isovalent

# PROBES

**probe**  **BPF program**

**attach layer**

**stats**

**lost 2nd bpf program**

```
=> kprobes_inc_nmissed_count
=> kprobe_ftrace_handler
=> 0xffffffffc05e00fb
=> __put_task_struct
```

**irq triggers rcu cleanup**

```
=> rcu_core
=> rcu_core_si
=> __do_softirq
=> __irq_exit_rcu
=> irq_exit_rcu
=> sysvec_apic_timer_interrupt
=> asm_sysvec_apic_timer_interrupt
=> native_write_msr
```

**1st bpf program**

```
=> x2apic_send_IPI_self
=> arch_irq_work_raise
=> __irq_work_queue_local
=> irq_work_queue
=> perf_output_put_handle
=> perf_output_end
=> perf_event_output
=> bpf_perf_event_output
=> bpf_prog_3f2efd9e3c2...
=> trace_call_bpf
=> kprobe_perf_func
=> kprobe_dispatcher
=> kprobe_ftrace_handler
=> wake_up_new_task
=> __do_sys_clone
=> __x64_sys_clone
=> do_syscall_64
=> entry_SYSCALL_64_after_hwframe
```

# STATS

## attach layer missed counts

```
# bpftool link

7: perf_event  prog 78
        kprobe ffffffffa039e910 bpf_kfunc_common_test  missed 1
```

## bpf program missed counts

```
# bpftool prog

192: kprobe  name test5  tag bcf7977d3b93787c  gpl recursion_misses 1
        loaded_at 2023-11-05T14:23:13+0100  uid 0
        xlated 16B  jited 15B  memlock 4096B
        btf_id 163
        pids test_progs(794)
```

# KPROBE

- perf event local or global (legacy)
- do SET_BPF ioctl on perf event OR
- use perf link
- ftrace/int3/opt flavors

```
kernel/trace/trace_kprobe.c:

  tk->rp.kp.pre_handler = kprobe_dispatcher;
  ...
  ret = register_kprobe(&tk->rp.kp)
```

## KPROBE FTRACE

- for entry points
- needs FTRACE ;-)
- compiled with $(CC_FLAGS_FTRACE)

```
__x64_sys_read:
  call   ffffffff810f8110 <__fentry__>
```

# FTRACE

```
kprobe_ftrace_handler
{

  bit = ftrace_test_recursion_trylock(ip, parent_ip);
  if (bit < 0)
          return;

  if (kprobe_running()) {
          kprobes_inc_nmissed_count(p);

  ...

  p->pre_handler(p, regs)

}
```

# KPROBE INT3/OPT

## everything else ;-)

```
kernel/locking/Makefile:CFLAGS_REMOVE_lockdep.o = $(CC_FLAGS_FTRACE)
kernel/locking/Makefile:CFLAGS_REMOVE_lockdep_proc.o = $(CC_FLAGS_FTRACE)
kernel/locking/Makefile:CFLAGS_REMOVE_mutex-debug.o = $(CC_FLAGS_FTRACE)

kernel/trace/Makefile:ccflags-remove-$(CONFIG_FUNCTION_TRACER) += $(CC_FLAGS_FTRACE)
kernel/trace/Makefile:CFLAGS_trace_selftest_dynamic.o = $(CC_FLAGS_FTRACE)
kernel/trace/Makefile:CFLAGS_trace_kprobe_selftest.o = $(CC_FLAGS_FTRACE)

lib/Makefile:ccflags-remove-$(CONFIG_FUNCTION_TRACER) += $(CC_FLAGS_FTRACE)
lib/Makefile:CFLAGS_test_fprobe.o += $(CC_FLAGS_FTRACE)

...
```

```
__x64_sys_read:
  ...
  int 3
```

# int 3 trap code

```
kprobe_int3_handler
{

  if (kprobe_running()) {
    if (reenter_kprobe(p, regs, kcb))
      return 1;

  ...

  p->pre_handler(p, regs)

}
```

```
__x64_sys_read:
  ...
  jmp trampoline
```

**opt trampoline**

```
optimized_callback
{

  if (kprobe_running()) {
    kprobes_inc_nmissed_count(&op->kp);
  } else {
    ...
    p->pre_handler(p, regs)
  }
}
```

# KPROBE TRACE

```
p->pre_handler(p, regs)
```

```
kprobe_dispatcher
  kprobe_perf_func
    trace_call_bpf
    {
      if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
         rcu_read_lock();
         bpf_prog_inc_misses_counters(rcu_dereference(call->prog_array));
         rcu_read_unlock();
         ret = 0;
         goto out;
       }
```

# RETURN KPROBE

- for entry points only
- rethook only on x86

```
kernel/trace/trace_kprobe.c:

  tk->rp.handler = kretprobe_dispatcher;
  ...
  ret = register_kretprobe(&tk->rp);
```

```
__x64_sys_read:

p->pre_handler(p, regs)

pre_handler_kretprobe
{
  rhn = rethook_try_get(rp->rh);
  if (!rhn) {
    rp->nmissed++;
    return 0;
  }

  rethook_hook(...
}

arch_rethook_trampoline
  arch_rethook_trampoline_callback
    rethook_trampoline_handler
      kretprobe_rethook_handler
      {
        ...
        rp->handler(ri, regs);
      }
```

# RETURN KPROBE TRACE

```
rp->handler(ri, regs)

kretprobe_dispatcher
  kretprobe_perf_func
    trace_call_bpf
    {
      if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
         rcu_read_lock();
         bpf_prog_inc_misses_counters(rcu_dereference(call->prog_array));
         rcu_read_unlock();
         ret = 0;
         goto out;
       }

       ...
     }
```

# KPROBE MULTI

## based on fprobe/ftrace

## kprobe_multi link

```
kernel/trace/bpf_trace.c:

  link->fp.entry_handler = kprobe_multi_link_handler;
  ...
  err = register_fprobe_ips(&link->fp, addrs, cnt);
```

```
__x64_sys_read:
  call   ffffffff810f8110 <__fentry__>
```

# FTRACE

```
fprobe_handler
{
  bit = ftrace_test_recursion_trylock(ip, parent_ip);
  if (bit < 0) {
    fp->nmissed++;
    return;
  }

  fp->entry_handler(...)

}
```

```
fp->entry_handler(...)
```

```
kprobe_multi_link_handler
{
  if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
    bpf_prog_inc_misses_counter(link->link.prog);
    err = 0;
    goto out;
  }

  err = bpf_prog_run(link->link.prog, regs);
  ...
}
```

# UPROBE/UPROBE_MULTI

## 64 nested uretprobes allowed

```
uprobe: omit uretprobe due to nestedness limit pid/tgid=1034/1034
```

## no re-entry checks on bpf layer

## no missed counts ;-)

# TRACEPOINT

## perf/raw flavors

```
trace_##call(…)
```

# TRACEPOINT

```
perf_trace_##call
{

  entry = perf_trace_buf_alloc(__entry_size, &__regs, &rctx);
  if (!entry)
    return;

  ...

  perf_trace_run_bpf_submit
    trace_call_bpfperf_trace_##call
      trace_call_bpf
      {
        if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
           rcu_read_lock();
           bpf_prog_inc_misses_counters(rcu_dereference(call->prog_array));
           ...

      }
```

```
trace_##call(…)
```

# TRACEPOINT

```
__bpf_trace_##call
  __bpf_trace_run
  {
    if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
      bpf_prog_inc_misses_counter(prog);
      goto out;

    ...

  }
```

# PERF EVENT

- HW/SW event
- no tracepoint, no kprobe, no uprobe

# NMI/SWEVENT

```
perf_event_overflow
  __perf_event_overflow
    bpf_overflow_handler
    {

      if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1))
        goto out;

      ret = bpf_prog_run(prog, &ctx);

     out:
       __this_cpu_dec(bpf_prog_active);
    }
```

## TRAMPOLINE

- entry points only
- attached through ftrace direct interface

```
__x64_sys_read:
  call trampoline
```

```
trampoline:
  ...
  call   __bpf_prog_enter_recur
```

```
if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
  bpf_prog_inc_misses_counter(prog);
  return 0;
}
```

```
  mov    %rax,%rbx
  test   %rax,%rax
  je     skip
  lea    -0x8(%rbp),%rdi
  call   bpf_prog

skip:

  call   __bpf_prog_exit_recur

  ...
  ret
```

# BPF RE-ENTRY CHECKS

## per-program

### more permissive, possible unexpected traces

```
if (unlikely(this_cpu_inc_return(*(prog->active)) != 1)) {
    DROP
}
```

## per-cpu

### blocks bpf programs run on cpu

### bpf_disable/enable_instrumentation

```
if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
    DROP
}
```

# BPF RE-ENTRY CHECKS

## per-program

- trampoline
- raw tracepoint

## per-cpu

- kprobe
- kprobe.multi
- perf tracepoint
- perf event

# BPF RE-ENTRY CHECKS

## per-program

- trampoline
- raw tracepoint
- kprobe.multi

## per-cpu

- kprobe
- ~~kprobe.multi~~
- perf tracepoint
- perf event

# KPROBE

## bpf-1 attached on wake_up_new_task

## bpf-2 attached on __put_task_struct

```
wake_up_new_task
{ kprobe

  irq
  {
    __put_task_struct
    { kprobe

    }
  }
}
```

```
bpf-1/bpf_prog_active=0/
      kprobe_running=0

bpf-2/bpf_prog_active=1/
      kprobe_running=1
```

EXECUTED

```
bpf-1
```

# KPROBE MULTI

## bpf-1 attached on wake_up_new_task

## bpf-2 attached on __put_task_struct

```
wake_up_new_task
{ kprobe

  irq
  {
    __put_task_struct
    { kprobe

    }
  }
}
```

```
bpf-1/bpf_prog_active=0/
      ctx[task]=0

bpf-2/bpf_prog_active=1/
      ctx[irq]=0
```

EXECUTED

```
bpf-1
```

# KPROBE MULTI with prog->active

## bpf-1 attached on wake_up_new_task

## bpf-2 attached on __put_task_struct

```
wake_up_new_task
{ kprobe

  irq
  {
    __put_task_struct
    { kprobe

    }
  }
}
```

```
bpf-1/active=0/
      ctx[task]=0

bpf-2/active=0/
      ctx[irq]=0
```

EXECUTED

```
bpf-1

bpf-2
```

# DO WE NEED MORE COUNTERS.. ?

## kprobes

```
func1 (link1/missed=X1)
                                  BPF prog/missed=Y
func2 (link2/missed=X2)
```

## kprobe multi

```
func1 (link/missed=X)
func2                             BPF prog/missed=Y
...
funcX
```

**thanks, questions..**