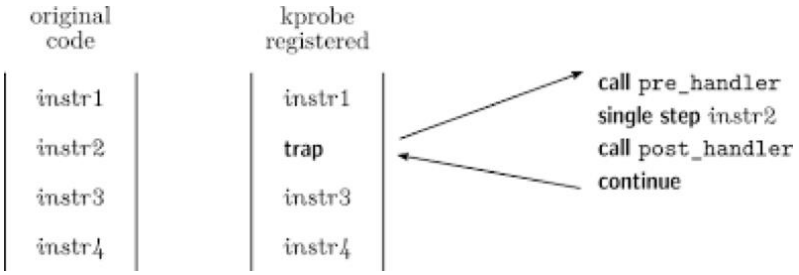
Kprobe 使用指南

kprobe 是 linux 内核的一个重要特性，是其他的内核调试工具（perf, systemtap）的“基础设施”，同时内核 BPF 也依赖 kprobe
它利用指令桩原理，截获指令流，并在指令执行前后插入 hook 函数:



如果需要知道内核函数是否被调用、被调用上下文、入参以及返回值，比较简单的方法是加 printk，但是效率低。
利用 kprobe 技术，用户可以自定义自己的回调函数，可以再几乎所有的函数中动态插入探测点。
当内核执行流程执行到指定的探测函数时, 会调用该回调函数, 用户即可收集所需的信息了，同时内核最后还会回到原本的正常执行流程。如果用户已经收集足够的信息，不再需要继续探测，则同样可以动态的移除探测点。
kprobes 技术包括的 2 种探测手段分别时 kprobe 和 kretprobe。
首先 kprobe 是最基本的探测方式，是实现后两种的基础，它可以在任意的位置放置探测点（就连函数内部的某条指令处也可以），它提供了探测点的调用前、调用后和内存访问出错 3 种回调方式，分别是 pre_handler、post_handler 和 fault_handler，其中 pre_handler 函数将在被探测指令被执行前回调，post_handler 会在被探测指令执行完毕后回调（注意不是被探测函数），fault_handler 会在内存访问出错时被调用；最后 kretprobe 从名字种就可以看出其用途了，它同样基于 kprobe 实现，用于获取被探测函数的返回值。

基本使用指南

开启内核：

```
Symbol: FTRACE [=y]
Type  : boolean
Prompt: Tracers
  Location:
(5) -> Kernel hacking
  Defined at kernel/trace/Kconfig:132
  Depends on: TRACING_SUPPORT [=y]
```

```
Symbol: KPROBE_EVENT [=y]
Type  : boolean
Prompt: Enable kprobes-based dynamic events
  Location:
    -> Kernel hacking
(1)   -> Tracers (FTRACE [=y])
  Defined at kernel/trace/Kconfig:405
  Depends on: TRACING_SUPPORT [=y] && FTRACE [=y] && KPROBES [=y] && HAVE_R
  Selects: TRACING [=y] && PROBE_EVENTS [=y]

Symbol: HAVE_KPROBES_ON_FTRACE [=y]
Type  : boolean
  Defined at arch/Kconfig:183
  Selected by: csky [=y]

Symbol: KPROBES_ON_FTRACE [=y]
Type  : boolean
  Defined at arch/Kconfig:79
  Depends on: KPROBES [=y] && HAVE_KPROBES_ON_FTRACE [=y] && DYNAMIC_FTRACE
```

终端运行:

首先通过 mount 获得 ftrace debug 接口，然后通过 kprobe_evets 注册你需要 probe 的内核函数，在 tracing/events/kprobes/<events>/ 下可以控制该 kprobe 函数的 开启和关闭

```
# mount -t debugfs nodev /sys/kernel/debug/


# echo 'p:myprobe _do_fork dfd=%a0 filename=%a1 flags=%a2 mode=+4($stack)'
# echo 'r:myretprobe _do_fork $retval' >> /sys/kernel/debug/tracing/kprobe_

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

# cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
# entries-in-buffer/entries-written: 8/8 #P:1
#
# _-----=> irqs-off
# / _----=> need-resched
```

```
# | / _---=> hardirq/softirq
# || / _--=> preempt-depth
# ||| / delay
# TASK-PID CPU# |||| TIMESTAMP FUNCTION
# | | | |||| | |
swapper/0-1 [000] dn.. 2.488544: Unknown type 599
swapper/0-1 [000] dn.. 2.489270: Unknown type 600
sh-121 [000] d... 408.113780: myprobe: (_do_fork+0x0/0x3ac) dfd=0xbe485f20
sh-121 [000] d... 408.117058: myretprobe: (sys_clone+0xa6/0xac <- _do_fork)
sh-121 [000] d... 409.816850: myprobe: (_do_fork+0x0/0x3ac) dfd=0xbe485f20
sh-121 [000] dn.. 409.817539: myretprobe: (sys_clone+0xa6/0xac <- _do_fork)
sh-121 [000] d... 411.202079: myprobe: (_do_fork+0x0/0x3ac) dfd=0xbe485f20
sh-121 [000] d... 411.202750: myretprobe: (sys_clone+0xa6/0xac <- _do_fork)
```

社区 kprobe 文档非常完善:

https://www.kernel.org/doc/Documentation/kprobes.txt

https://lwn.net/Articles/410200/

Trace-cmd 和 kernelshark:

Trace-cmd 是一个基于 ftrace 的用户态前端命令行工具，它的仓库在:

git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git

很多发行版，都有 trace-cmd 的包，完善的 man 手册

具体使用参考:

https://lwn.net/Articles/410200/