第三届 eBPF开发者大会
www.ebpftravel.com

深挖eBPF的插桩和回调过程

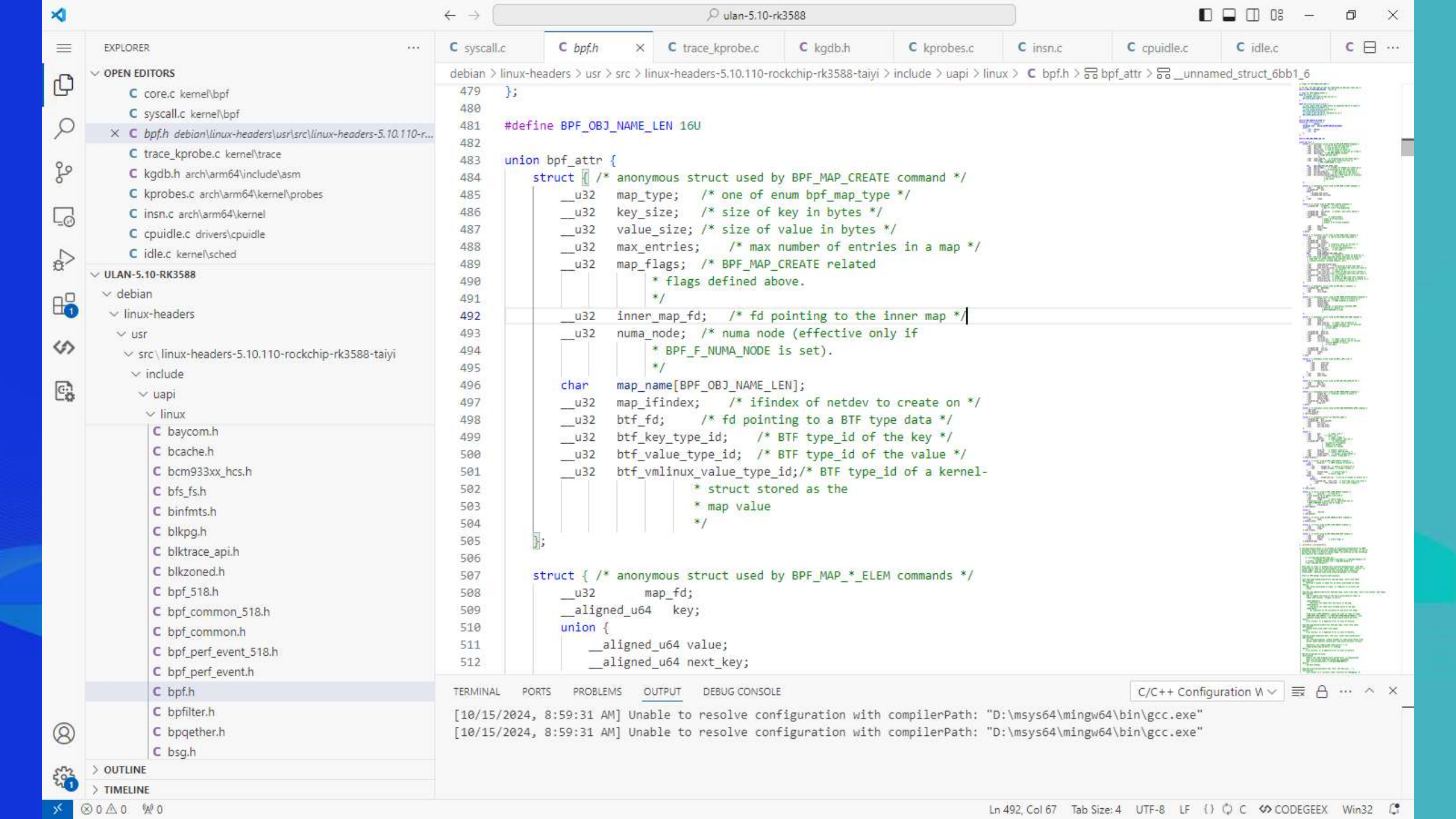
张银奎 《软件调试》作者

2025-4-19

中国·西安

系统调用 → 修补内核 → 触发回调 → 输出事件 → 调试方法

```
SYSCALL_DEFINE3(bpf, int, cmd, union bpf_attr
__user *, uattr, unsigned int, size)
```


ulan-5.10-rk3588
EXPLORER
OPEN EDITORS
core.c kernel\bpf
syscall.c kernel\bpf
bpf.h debian\linux-headers\usr\src\linux-headers-5.10.110-r...
trace_kprobe.c kernel\trace
kgdb.h arch\arm64\include\asm
kprobes.c arch\arm64\kernel\probes
insn.c arch\arm64\kernel
cpuidle.c drivers\cpuidle
idle.c kernel\sched
ULAN-5.10-RK3588
debian
linux-headers
usr
src\linux-headers-5.10.110-rockchip-rk3588-taiyi
include
uapi
linux
baycom.h
bcache.h
bcm933xx_hcs.h
bfs_fs.h
binfmts.h
blkpg.h
blktrace_api.h
blkzoned.h
bpf_518.h
bpf_common_518.h
bpf_common.h
bpf_perf_event_518.h
bpf_perf_event.h
bpf.h
bpfilter.h
bpqether.h
bsg.h
OUTLINE
TIMELINE
syscall.c
bpf.h
trace_kprobe.c
kgdb.h
kprobes.c
insn.c
cpuidle.c
idle.c
debian > linux-headers > usr > src > linux-headers-5.10.110-rockchip-rk3588-taiyi > include > uapi > linux > bpf.h > bpf_attr > __unnamed_struct_6bb1_6
479 };
480
481 #define BPF_OBJ_NAME_LEN 16U
482
483 union bpf_attr {
484 struct { /* anonymous struct used by BPF_MAP_CREATE command */
485 __u32 map_type; /* one of enum bpf_map_type */
486 __u32 key_size; /* size of key in bytes */
487 __u32 value_size; /* size of value in bytes */
488 __u32 max_entries; /* max number of entries in a map */
489 __u32 map_flags; /* BPF_MAP_CREATE related
490 * flags defined above.
491 */
492 __u32 inner_map_fd; /* fd pointing to the inner map */
493 __u32 numa_node; /* numa node (effective only if
494 * BPF_F_NUMA_NODE is set).
495 */
496 char map_name[BPF_OBJ_NAME_LEN];
497 __u32 map_ifindex; /* ifindex of netdev to create on */
498 __u32 btf_fd; /* fd pointing to a BTF type data */
499 __u32 btf_key_type_id; /* BTF type_id of the key */
500 __u32 btf_value_type_id; /* BTF type_id of the value */
501 __u32 btf_vmlinux_value_type_id;/* BTF type_id of a kernel-
502 * struct stored as the
503 * map value
504 */
505 };
506
507 struct { /* anonymous struct used by BPF_MAP_*_ELEM commands */
508 __u32 map_fd;
509 __aligned_u64 key;
510 union {
511 __aligned_u64 value;
512 __aligned_u64 next_key;
TERMINAL
PORTS
PROBLEMS
OUTPUT
DEBUG CONSOLE
C/C++ Configuration
[10/15/2024, 8:59:31 AM] Unable to resolve configuration with compilerPath: "D:\msys64\mingw64\bin\gcc.exe"
[10/15/2024, 8:59:31 AM] Unable to resolve configuration with compilerPath: "D:\msys64\mingw64\bin\gcc.exe"
Ln 492, Col 67
Tab Size: 4
UTF-8
LF
C
CODEGEEX
Win32

```
geduer@ulan:~$ cat HelloYourLand.py
#!/usr/bin/python

from bcc import BPF

prog = """
int hello(void *ctx){
    bpf_trace_printk("Here is YourLand!\\n");
    return 0;
}
"""
b = BPF(text=prog)
b.attach_kprobe(event=b.get_syscall_fnname("clone"), fn_name="hello")

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)","COMM","PID","MESSAGE"))
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        continue
    print("%-18.9f %-16s %-d %s" % (ts,task,pid,msg))g
```

```
geduer@ulan:~$ sudo ./HelloYourLand.py
[sudo] password for geduer:
TIME(s)            COMM             PID    MESSAGE
359.216558000      b'<...>'         2759 b'Here is YourLand!'
365.169917000      b'<...>'         2588 b'Here is YourLand!'
365.182842000      b'<...>'         2812 b'Here is YourLand!'
366.049768000      b'<...>'         1 b'Here is YourLand!'
366.328557000      b'sshd'          2588 b'Here is YourLand!'
366.349306000      b'sshd'          2816 b'Here is YourLand!'
366.355594000      b'<...>'         2817 b'Here is YourLand!'
366.367741000      b'<...>'         2819 b'Here is YourLand!'
366.375037000      b'<...>'         2820 b'Here is YourLand!'
366.391117000      b'00-header'     2820 b'Here is YourLand!'
366.399792000      b'00-header'     2820 b'Here is YourLand!'
366.407825000      b'00-header'     2820 b'Here is YourLand!'
366.418288000      b'run-parts'     2819 b'Here is YourLand!'
366.427935000      b'run-parts'     2819 b'Here is YourLand!'
366.433061000      b'run-parts'     2819 b'Here is YourLand!'
366.437419000      b'85-fwupd'      2827 b'Here is YourLand!'
366.445348000      b'run-parts'     2819 b'Here is YourLand!'
366.452698000      b'90-updates-avai' 2829 b'Here is YourLand!'
366.459509000      b'90-updates-avai' 2829 b'Here is YourLand!'
366.461085000      b'90-updates-avai' 2829 b'Here is YourLand!'
366.473640000      b'run-parts'     2819 b'Here is YourLand!'
```

```
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=2, insns=0x7fffca1100,
license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_version=KERNEL_VERSION(0, 0, 0),
prog_flags=0, prog_name="libbpf_nametest", prog_ifindex=0,
expected_attach_type=BPF_CGROUP_INET_INGRESS, prog_btf_fd=0, func_info_rec_size=0,
func_info=NULL, func_info_cnt=0, line_info_rec_size=0, line_info=NULL, line_info_cnt=0, attach_btf_id=0,
attach_prog_fd=0, fd_array=NULL}, 148) = 4
close(4)                                = 0
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_KPROBE, insn_cnt=16, insns=0x7f91e18000,
license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_version=KERNEL_VERSION(5, 10, 110),
prog_flags=0, prog_name="hello", prog_ifindex=0, expected_attach_type=BPF_CGROUP_INET_INGRESS,
prog_btf_fd=3, func_info_rec_size=8, func_info=0x48bc550, func_info_cnt=1, line_info_rec_size=16,
line_info=0x591fe30, line_info_cnt=5, attach_btf_id=0, attach_prog_fd=0, fd_array=NULL}, 148) = 4
openat(AT_FDCWD, "/sys/bus/event_source/devices/kprobe/type", O_RDONLY) = 5
read(5, "6\n", 4096)                    = 2
close(5)                                = 0
openat(AT_FDCWD, "/sys/bus/event_source/devices/kprobe/format/retprobe", O_RDONLY) = 5
read(5, "config:0\n", 4096)             = 9
close(5)                                = 0
perf_event_open({type=0x6 /* PERF_TYPE_??? */, size=0x88 /* PERF_ATTR_SIZE_??? */, config=0,
sample_period=1, sample_type=0, read_format=0, precise_ip=0 /* arbitrary skid */, ...}, -1, 0, -1,
PERF_FLAG_FD_CLOEXEC) = 5
ioctl(5, PERF_EVENT_IOC_SET_BPF, 4)     = 0
ioctl(5, PERF_EVENT_IOC_ENABLE, 0)      = 0
```

```
write(1, "TIME(s)          COMM        "..., 51TIME(s)          COMM           PID    MESSAGE
) = 51
openat(AT_FDCWD, "/sys/kernel/debug/tracing/trace_pipe", O_RDONLY|O_CLOEXEC) = 6
fstat(6, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
ioctl(6, TCGETS, 0x7fffca1c20)          = -1 ENOTTY (Inappropriate ioctl for device)
lseek(6, 0, SEEK_CUR)                   = -1 ESPIPE (Illegal seek)
read(6, 0x4f75f10, 4096)                = -1 EINTR (Interrupted system call)
```

系统调用 → 修补内核 → 触发回调 → 输出事件 → 调试方法

修补之前的__arm64_sys_clone

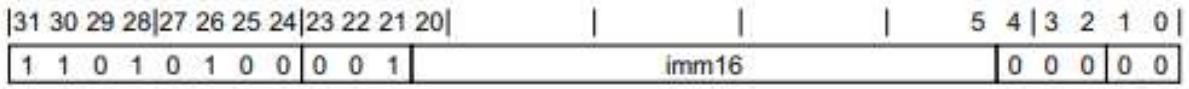
```
u ffffffc0`10096100
lk!__arm64_sys_clone [kernel/fork.c @ 2643]:
ffffffc010096100 d503245f hint  #0x22
ffffffc010096104 aa1e03e9 mov   x9, x30
ffffffc010096108 d503201f nop
ffffffc01009610c aa0003e4 mov   x4, x0
ffffffc010096110 d503233f paciasp
ffffffc010096114 a9bf7bfd stp   x29, x30, [sp, #-0x10]!
ffffffc010096118 910003fd mov   x29, sp
ffffffc01009611c a9400400 ldp   x0, x1, [x0]
```

修补后的__arm64_sys_clone

```
u ffffffc0`10096100
lk!__arm64_sys_clone [kernel/fork.c @ 2643]:

ffffffc010096100 d4200080 brk   #4
ffffffc010096104 aa1e03e9 mov   x9, x30
ffffffc010096108 d503201f nop
ffffffc01009610c aa0003e4 mov   x4, x0
ffffffc010096110 d503233f paciasp
ffffffc010096114 a9bf7bfd stp   x29, x30, [sp, #-0x10]!
ffffffc010096118 910003fd mov   x29, sp
ffffffc01009611c a9400400 ldp   x0, x1, [x0]
```

C6.2.38 BRK

Breakpoint instruction. A BRK instruction generates a Breakpoint Instruction exception. The PE records the exception in ESR_ELx, using the EC value 0x3c, and captures the value of the immediate argument in ESR_ELx.ISS.



Encoding

BRK #<imm>

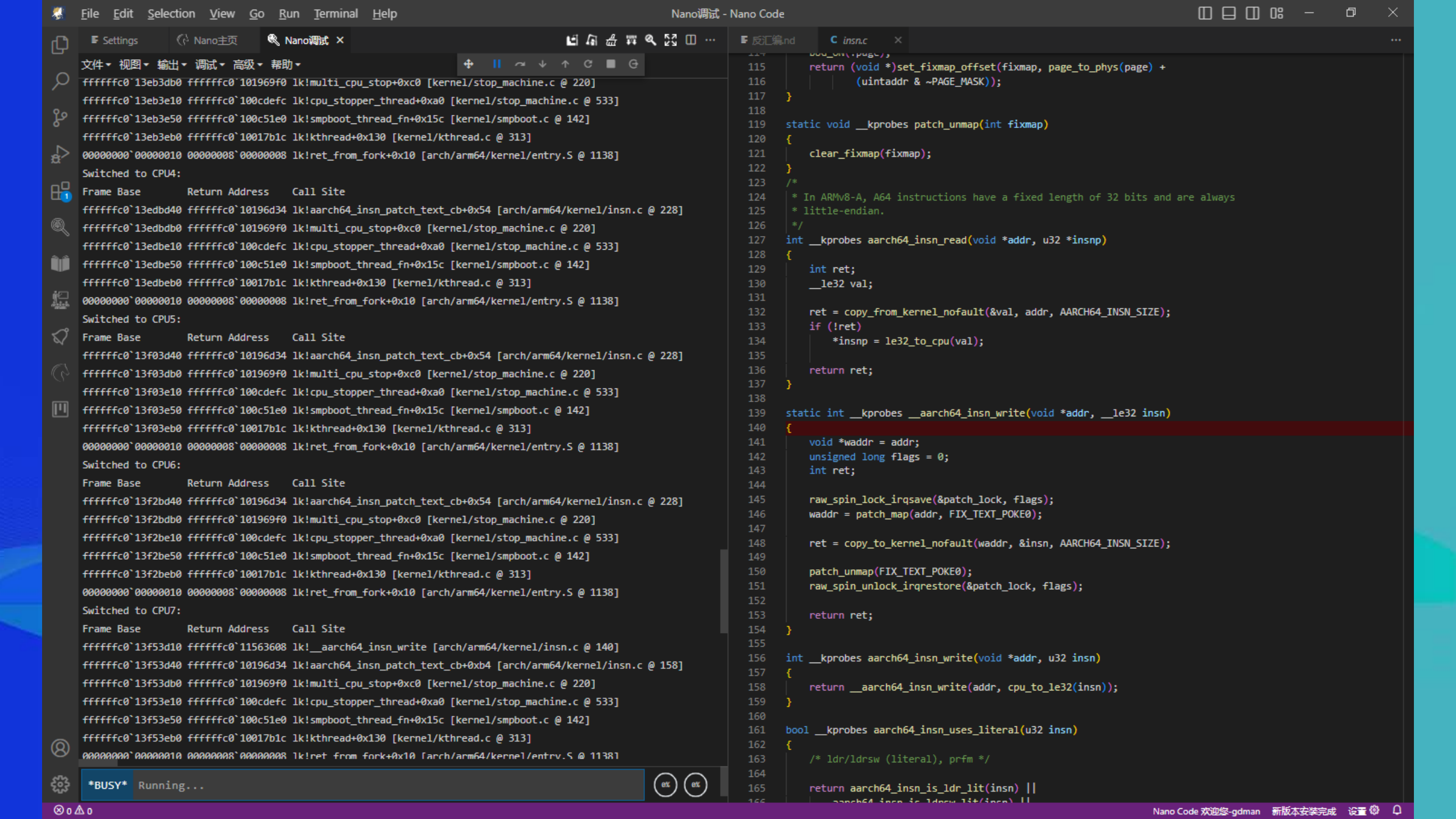
arch > arm64 > include > asm > C brk-imm.h

```
/*
 * #imm16 values used for BRK instruction generation
 * 0x004: for installing kprobes
 * 0x005: for installing uprobes
 * 0x006: for kprobe software single-step
 * Allowed values for kgdb are 0x400 - 0x7ff
 * 0x100: for triggering a fault on purpose (reserved)
 * 0x400: for dynamic BRK instruction
 * 0x401: for compile time BRK instruction
 * 0x800: kernel-mode BUG() and WARN() traps
 * 0x9xx: tag-based KASAN trap (allowed values 0x900 - 0x9ff)
 */
#define KPROBES_BRK_IMM                 0x004
#define UPROBES_BRK_IMM                 0x005
#define KPROBES_BRK_SS_IMM              0x006
#define FAULT_BRK_IMM                   0x100
#define KGDB_DYN_DBG_BRK_IMM            0x400
#define KGDB_COMPILED_DBG_BRK_IMM       0x401
#define BUG_BRK_IMM                     0x800
#define KASAN_BRK_IMM                   0x900
#define KASAN_BRK_MASK                  0x0ff
```

```
ffffffc010096100 d4200080 brk #4
```

```
static int __kprobes __aarch64_insn_write(void *addr, __le32 insn)
{
        void *waddr = addr;
        unsigned long flags = 0;
        int ret;

        raw_spin_lock_irqsave(&patch_lock, flags);
        waddr = patch_map(addr, FIX_TEXT_POKE0);

        ret = copy_to_kernel_nofault(waddr, &insn, AARCH64_INSN_SIZE);

        patch_unmap(FIX_TEXT_POKE0);
        raw_spin_unlock_irqrestore(&patch_lock, flags);

        return ret;
}
```

```
kn
 # Frame Base        Return Address    Call Site
00 ffffffc0`1761bad0 ffffffc0`11563f64 lk!aarch64_insn_patch_text [arch/arm64/kernel/insn.c @ 236]
01 ffffffc0`1761bb10 ffffffc0`101a8e5c lk!arch_prepare_kprobe+0xa4 [arch/arm64/kernel/probes/kprobes.c
@ 48]
02 ffffffc0`1761bb70 ffffffc0`101fd7dc lk!register_kprobe+0x41c [kernel/kprobes.c @ 1721]
03 ffffffc0`1761bba0 ffffffc0`101ff7c0 lk!__register_trace_kprobe+0x108 [kernel/trace/trace_kprobe.c @
514]
04 ffffffc0`1761bbd0 ffffffc0`101f1d4c lk!create_local_trace_kprobe+0xc0 [kernel/trace/trace_kprobe.c
@ 1841]
05 ffffffc0`1761bc10 ffffffc0`10258bb0 lk!perf_kprobe_init+0x8c [kernel/trace/trace_event_perf.c @ 275]
06 ffffffc0`1761bc30 ffffffc0`10259924 lk!perf_kprobe_event_init+0x50 [kernel/events/core.c @ 9619]
07 ffffffc0`1761bc60 ffffffc0`1025d2b8 lk!perf_try_init_event+0x54 [kernel/events/core.c @ 11001]
08 ffffffc0`1761bd00 ffffffc0`10264580 lk!perf_event_alloc+0x384 [kernel/events/core.c @ 11052]
09 ffffffc0`1761be20 ffffffc0`1026518c lk!__do_sys_perf_event_open+0x180 [kernel/events/core.c @ 11808]
0a ffffffc0`1761be30 ffffffc0`1002b668 lk!__arm64_sys_perf_event_open+0x2c [kernel/events/core.c @
11698]
0b ffffffc0`1761be70 ffffffc0`1002b8ec lk!el0_svc_common.constprop.0+0xa8 [arch/arm64/kernel/syscall.c
@ 36]
0c ffffffc0`1761be80 ffffffc0`11556170 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
0d ffffffc0`1761bea0 ffffffc0`11556c44 lk!el0_svc+0x20 [arch/arm64/kernel/entry-common.c @ 358]
0e ffffffc0`1761beb0 ffffffc0`100127e0 lk!el0_sync_handler+0xa4 [arch/arm64/kernel/entry-common.c @
374]
0f ffffffc0`1761bff0 00000000`00000000 lk!el0_sync+0x1a0 [arch/arm64/kernel/entry.S @ 789]
```


File Edit Selection View Go Run Terminal Help
Nano调试 - Nano Code
Settings
Nano主页
Nano调试
文件 视图 输出 调试 高级 帮助
反汇编.nd
insn.c
ffffffc0`13eb3db0 ffffffc0`101969f0 lk!multi_cpu_stop+0xc0 [kernel/stop_machine.c @ 220]
ffffffc0`13eb3e10 ffffffc0`100cdefc lk!cpu_stopper_thread+0xa0 [kernel/stop_machine.c @ 533]
ffffffc0`13eb3e50 ffffffc0`100c51e0 lk!smpboot_thread_fn+0x15c [kernel/smpboot.c @ 142]
ffffffc0`13eb3eb0 ffffffc0`10017b1c lk!kthread+0x130 [kernel/kthread.c @ 313]
00000000`00000010 00000008`00000008 lk!ret_from_fork+0x10 [arch/arm64/kernel/entry.S @ 1138]
Switched to CPU4:
Frame Base Return Address Call Site
ffffffc0`13edbd40 ffffffc0`10196d34 lk!aarch64_insn_patch_text_cb+0x54 [arch/arm64/kernel/insn.c @ 228]
ffffffc0`13edbdb0 ffffffc0`101969f0 lk!multi_cpu_stop+0xc0 [kernel/stop_machine.c @ 220]
ffffffc0`13edbe10 ffffffc0`100cdefc lk!cpu_stopper_thread+0xa0 [kernel/stop_machine.c @ 533]
ffffffc0`13edbe50 ffffffc0`100c51e0 lk!smpboot_thread_fn+0x15c [kernel/smpboot.c @ 142]
ffffffc0`13edbeb0 ffffffc0`10017b1c lk!kthread+0x130 [kernel/kthread.c @ 313]
00000000`00000010 00000008`00000008 lk!ret_from_fork+0x10 [arch/arm64/kernel/entry.S @ 1138]
Switched to CPU5:
Frame Base Return Address Call Site
ffffffc0`13f03d40 ffffffc0`10196d34 lk!aarch64_insn_patch_text_cb+0x54 [arch/arm64/kernel/insn.c @ 228]
ffffffc0`13f03db0 ffffffc0`101969f0 lk!multi_cpu_stop+0xc0 [kernel/stop_machine.c @ 220]
ffffffc0`13f03e10 ffffffc0`100cdefc lk!cpu_stopper_thread+0xa0 [kernel/stop_machine.c @ 533]
ffffffc0`13f03e50 ffffffc0`100c51e0 lk!smpboot_thread_fn+0x15c [kernel/smpboot.c @ 142]
ffffffc0`13f03eb0 ffffffc0`10017b1c lk!kthread+0x130 [kernel/kthread.c @ 313]
00000000`00000010 00000008`00000008 lk!ret_from_fork+0x10 [arch/arm64/kernel/entry.S @ 1138]
Switched to CPU6:
Frame Base Return Address Call Site
ffffffc0`13f2bd40 ffffffc0`10196d34 lk!aarch64_insn_patch_text_cb+0x54 [arch/arm64/kernel/insn.c @ 228]
ffffffc0`13f2bdb0 ffffffc0`101969f0 lk!multi_cpu_stop+0xc0 [kernel/stop_machine.c @ 220]
ffffffc0`13f2be10 ffffffc0`100cdefc lk!cpu_stopper_thread+0xa0 [kernel/stop_machine.c @ 533]
ffffffc0`13f2be50 ffffffc0`100c51e0 lk!smpboot_thread_fn+0x15c [kernel/smpboot.c @ 142]
ffffffc0`13f2beb0 ffffffc0`10017b1c lk!kthread+0x130 [kernel/kthread.c @ 313]
00000000`00000010 00000008`00000008 lk!ret_from_fork+0x10 [arch/arm64/kernel/entry.S @ 1138]
Switched to CPU7:
Frame Base Return Address Call Site
ffffffc0`13f53d10 ffffffc0`11563608 lk!__aarch64_insn_write [arch/arm64/kernel/insn.c @ 140]
ffffffc0`13f53d40 ffffffc0`10196d34 lk!aarch64_insn_patch_text_cb+0xb4 [arch/arm64/kernel/insn.c @ 158]
ffffffc0`13f53db0 ffffffc0`101969f0 lk!multi_cpu_stop+0xc0 [kernel/stop_machine.c @ 220]
ffffffc0`13f53e10 ffffffc0`100cdefc lk!cpu_stopper_thread+0xa0 [kernel/stop_machine.c @ 533]
ffffffc0`13f53e50 ffffffc0`100c51e0 lk!smpboot_thread_fn+0x15c [kernel/smpboot.c @ 142]
ffffffc0`13f53eb0 ffffffc0`10017b1c lk!kthread+0x130 [kernel/kthread.c @ 313]
BUSY Running...
return (void *)set_fixmap_offset(fixmap, page_to_phys(page) +
(uintaddr & ~PAGE_MASK));
}
static void __kprobes patch_unmap(int fixmap)
{
clear_fixmap(fixmap);
}
/*
* In ARMv8-A, A64 instructions have a fixed length of 32 bits and are always
* little-endian.
*/
int __kprobes aarch64_insn_read(void *addr, u32 *insnp)
{
int ret;
__le32 val;
ret = copy_from_kernel_nofault(&val, addr, AARCH64_INSN_SIZE);
if (!ret)
*insnp = le32_to_cpu(val);
return ret;
}
static int __kprobes __aarch64_insn_write(void *addr, __le32 insn)
{
void *waddr = addr;
unsigned long flags = 0;
int ret;
raw_spin_lock_irqsave(&patch_lock, flags);
waddr = patch_map(addr, FIX_TEXT_POKE0);
ret = copy_to_kernel_nofault(waddr, &insn, AARCH64_INSN_SIZE);
patch_unmap(FIX_TEXT_POKE0);
raw_spin_unlock_irqrestore(&patch_lock, flags);
return ret;
}
int __kprobes aarch64_insn_write(void *addr, u32 insn)
{
return __aarch64_insn_write(addr, cpu_to_le32(insn));
}
bool __kprobes aarch64_insn_uses_literal(u32 insn)
{
/* ldr/ldrsw (literal), prfm */
return aarch64_insn_is_ldr_lit(insn) ||
Nano Code 欢迎您-gdman
新版本安装完成
设置

```
dt lk!aarch64_insn_patch 0xffffffc01761bab0
   +0x000 text_addrs        : 0xffffffc0`11563928  ->
0xd5384103`a9bd7bfd
   +0x008 new_insns         : 0x00000000`d503233f  -> ??
   +0x010 insn_cnt          : 2
   +0x014 cpu_count         : 8

 dd 0xffffffc01761bab0
 ffffffc0`1761bab0  11563928 ffffffc0 d503233f 00000000
 ffffffc0`1761bac0  00000002 00000008 f3bd1600 f106c3c9
 ffffffc0`1761bad0  1761bb10 ffffffc0 101a8e5c ffffffc0
 ffffffc0`1761bae0  4fe5ca18 ffffff81 00000000 00000000
 ffffffc0`1761baf0  d503245f d42000c0 13e9b000 ffffffc0
 ffffffc0`1761bb00  13e9b004 ffffffc0 f3bd1600 f106c3c9
 ffffffc0`1761bb10  1761bb70 ffffffc0 101fd7dc ffffffc0
 ffffffc0`1761bb20  00000000 00000000 4fe5ca00 ffffff81..
```

CPU4 Master

```
kn
 # Frame Base         Return Address    Call Site
00 ffffffc0`162f1900 ffffffc0`102bbd60 lk!vmacache_find+0x34 [mm/vmacache.c @ 32]
01 ffffffc0`162f1930 ffffffc0`11564c14 lk!find_vma+0x2c [mm/mmap.c @ 2411]
02 ffffffc0`162f19a0 ffffffc0`11564f40 lk!do_page_fault+0x224 [arch/arm64/mm/fault.c @ 608]
03 ffffffc0`162f19c0 ffffffc0`1003a770 lk!do_translation_fault+0xbc [arch/arm64/mm/fault.c @ 703]
04 ffffffc0`162f1a00 ffffffc0`115557b8 lk!do_mem_abort+0x4c [arch/arm64/mm/fault.c @ 837]
05 ffffffc0`162f1a40 ffffffc0`11556110 lk!el1_abort+0x98 [arch/arm64/kernel/entry-common.c @ 123]
06 ffffffc0`162f1a50 ffffffc0`10012408 lk!el1_sync_handler+0xb0 [arch/arm64/kernel/entry-common.c @
07 ffffffc0`162f1b90 ffffffc0`1016d330 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
08 ffffffc0`162f1ab0 ffffff81`0b4a5c40 lk!exit_robust_list+0x80 [./arch/arm64/include/asm/uaccess.h
```

Ready tasks of CPU 1:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	16	0	0	24893463	0	migration/1
0	00000000	2638	120	0	511561172	0	sshd

Ready tasks of CPU 2:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	21	0	0	30923962	0	migration/2

Ready tasks of CPU 3:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	26	0	0	23263629	0	migration/3

Ready tasks of CPU 4:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	31	0	0	4458709	0	migration/4

Ready tasks of CPU 5:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	36	0	0	5118461	0	migration/5

Ready tasks of CPU 6:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	41	0	0	3778248	0	migration/6

Ready tasks of CPU 7:

on_cpu	state	PID	prio	wait-time	sum-exec	summ-sleep	comm
1	00000000	46	0	0	2650080	0	migration/7
0	00000000	2585	120	0	3353868545	0	HelloYourLand.p

```
bl
 0 e ffffffc0`111b400c e 4 0001 (0001) lk!copy_bpf_fprog_from_user+c
 1 e ffffffc0`111b7f8c e 4 0001 (0001) lk!bpf_prog_create+c
 2 e ffffffc0`10201ca0 e 4 0001 (0001) lk!bpf_jit_compile+c
 3 e ffffffc0`101ffabc e 4 0001 (0001) lk!bpf_patch_insn_single+c
```

系统调用 → 修补内核 → 触发回调 → 输出事件 → 调试方法

```
int copy_bpf_fprog_from_user(struct sock_fprog *dst, sockptr_t src, int len);
```

```
int bpf_prog_create(struct bpf_prog **pfp, struct sock_fprog_kern *fprog);

int bpf_prog_create_from_user(struct bpf_prog **pfp, struct sock_fprog *fprog,
                              bpf_aux_classic_check_t trans, bool save_orig);

struct bpf_prog *bpf_int_jit_compile(struct bpf_prog *prog);
void bpf_jit_compile(struct bpf_prog *prog);
bool bpf_jit_needs_zext(void);


const char *__bpf_address_lookup(unsigned long addr, unsigned long *size,
                                 unsigned long *off, char *sym);
bool is_bpf_text_address(unsigned long addr);
int bpf_get_kallsym(unsigned int symnum, unsigned long *value, char *type,
                    char *sym);
```

```
k
Frame Base          Return Address    Call Site
ffffffc0`16bcbcb0 ffffffc0`10208268 lk!bpf_prog_load+0x8 [kernel/bpf/syscall.c @ 2123]
ffffffc0`16bcbe20 ffffffc0`10209e20 lk!__do_sys_bpf+0x6b8 [kernel/bpf/syscall.c @ 4428]
ffffffc0`16bcbe30 ffffffc0`1002b668 lk!__arm64_sys_bpf+0x2c [kernel/bpf/syscall.c @ 4383]
ffffffc0`16bcbe70 ffffffc0`1002b8ec lk!el0_svc_common.constprop.0+0xa8 [arch/arm64/kernel/syscall.c @ 36]
ffffffc0`16bcbe80 ffffffc0`11547810 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
ffffffc0`16bcbe90 ffffffc0`115482e4 lk!enter_from_user_mode+0x10 [arch/arm64/kernel/entry-common.c @ 238]
ffffffc0`16bcbea0 0000007f`99324e44 lk!el0_sync_handler+0x74 [arch/arm64/kernel/entry-common.c @ 416]
Loading symbols for 0000007f`99300000  libbpf.so.1.2.2 ->   Failed to load symbols for libbpf.so.1.2.2
Loading symbols for 0000007f`99300000  libbpf.so.1.2.2 ->   Failed to load symbols for libbpf.so.1.2.2
```

```
struct bpf_prog {
        u16                             pages;                  /* Number of allocated pages */
        u16                             jited:1,    /* Is our filter JIT'ed? */
                                        jit_requested:1,/* archs need to JIT the prog */
                                        gpl_compatible:1, /* Is filter GPL compatible? */
                                        cb_access:1,            /* Is control block accessed? */
                                        dst_needed:1,           /* Do we need dst entry? */
                                        blinded:1, /* Was blinded */
                                        is_func:1, /* program is a bpf function */
                                        kprobe_override:1, /* Do we override a kprobe? */
                                        has_callchain_buf:1, /* callchain buffer allocated? */
                                        enforce_expected_attach_type:1, /* Enforce expected_attach_type checking at attach time
                                        call_get_stack:1; /* Do we call bpf_get_stack() or bpf_get_stackid() */
        enum bpf_prog_type              type;                   /* Type of BPF program */
        enum bpf_attach_type            expected_attach_type; /* For some prog types */
        u32                             len;                    /* Number of filter blocks */
        u32                             jited_len;  /* Size of jited insns in bytes */
        u8                              tag[BPF_TAG_SIZE];
        struct bpf_prog_aux *aux;                       /* Auxiliary fields */
        struct sock_fprog_kern          *orig_prog;             /* Original BPF program */
        unsigned int                    (*bpf_func)(const void *ctx,
                                                    const struct bpf_insn *insn);

        /* Instructions for interpreter */
        struct sock_filter      insns[0];
        struct bpf_insn                 insnsi[];
};
```

```
#define BPF_PROG_RUN(prog, ctx)                                         \
        __BPF_PROG_RUN(prog, ctx, bpf_dispatcher_nop_func)

#define __BPF_PROG_RUN(prog, ctx, dfunc)        ({                      \
        u32 __ret;                                                      \
        cant_migrate();                                                 \
        if (static_branch_unlikely(&bpf_stats_enabled_key)) {           \
                struct bpf_prog_stats *__stats;                         \
                u64 __start = sched_clock();                            \
                __ret = dfunc(ctx, (prog)->insnsi, (prog)->bpf_func);   \
                __stats = this_cpu_ptr(prog->aux->stats);               \
                u64_stats_update_begin(&__stats->syncp);                \
                __stats->cnt++;                                         \
                __stats->nsecs += sched_clock() - __start;              \
                u64_stats_update_end(&__stats->syncp);                  \
        } else {                                                        \
                __ret = dfunc(ctx, (prog)->insnsi, (prog)->bpf_func);   \
        }                                                               \
        __ret; })

#define BPF_PROG_RUN(prog, ctx)                                         \
        __BPF_PROG_RUN(prog, ctx, bpf_dispatcher_nop_func)
```

```
kn
 # Frame Base         Return Address    Call Site
00 ffffffc0`1803bb20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
01 ffffffc0`1803bbb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
02 ffffffc0`1803bbe0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
03 ffffffc0`1803bc30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8
[arch/arm64/kernel/probes/kprobes.c @ 353]
04 ffffffc0`1803bc40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-
monitors.c @ 326]
05 ffffffc0`1803bc60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
06 ffffffc0`1803bca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
07 ffffffc0`1803bcd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
08 ffffffc0`1803bce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @ 223]
09 ffffffc0`1803be20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
0a ffffffc0`1803be40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0b ffffffc0`1803bcf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
0c ffffffc0`1803bec0 0000007f`f6408b70 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
```

File Edit Selection View Go Run Terminal Help — Nano调试 - Nano Code

Settings | Nano主页 | Nano调试 ×

文件 视图 输出 调试 高级 帮助

```
lk!trace_call_bpf:
ffffffc0101facf0 d503233f paciasp
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int

k
Frame Base        Return Address    Call Site
ffffffc0`17733b20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
ffffffc0`17733bb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
ffffffc0`17733be0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
ffffffc0`17733c30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8 [arch/arm64/kernel/probes/kprobes.c @ 353]
ffffffc0`17733c40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-monitors.c @ 326]
ffffffc0`17733c60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
ffffffc0`17733ca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
ffffffc0`17733cd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
ffffffc0`17733ce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @ 223]
ffffffc0`17733e20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
ffffffc0`17733e40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
ffffffc0`17733cf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
ffffffc0`17733ec0 00000000`00000000 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int

kn
 # Frame Base        Return Address    Call Site
00 ffffffc0`17733b20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
01 ffffffc0`17733bb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
02 ffffffc0`17733be0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
03 ffffffc0`17733c30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8 [arch/arm64/kernel/probes/kprobes.c @ 353]
04 ffffffc0`17733c40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-monitors.c @ 326]
05 ffffffc0`17733c60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
06 ffffffc0`17733ca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
07 ffffffc0`17733cd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
08 ffffffc0`17733ce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @ 223]
09 ffffffc0`17733e20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
0a ffffffc0`17733e40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0b ffffffc0`17733cf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
0c ffffffc0`17733ec0 00000000`00000000 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int
```

6: kd>

entry.S | process.c | bpf_trace.c × | 反汇编.nd

```
86  * Return: BPF programs always return an integer which is interpreted by
87  * kprobe handler as:
88  * 0 - return from kprobe (event is filtered out)
89  * 1 - store kprobe event into ring buffer
90  * Other values are reserved and currently alias to 1
91  */
92 unsigned int trace_call_bpf(struct trace_event_call *call, void *ctx)
93 {
94     unsigned int ret;
95
96     cant_sleep();
97
98     if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
99         /*
100         * since some bpf program is already running on this cpu,
101         * don't call into another bpf program (same or different)
102         * and don't send kprobe event into ring-buffer,
103         * so return zero here
104         */
105        ret = 0;
106        goto out;
107    }
108
109    /*
110     * Instead of moving rcu_read_lock/rcu_dereference/rcu_read_unlock
111     * to all call sites, we did a bpf_prog_array_valid() there to check
112     * whether call->prog_array is empty or not, which is
113     * a heurisitc to speed up execution.
114     *
115     * If bpf_prog_array_valid() fetched prog_array was
116     * non-NULL, we go into trace_call_bpf() and do the actual
117     * proper rcu_dereference() under RCU lock.
118     * If it turns out that prog_array is NULL then, we bail out.
119     * For the opposite, if the bpf_prog_array_valid() fetched pointer
120     * was NULL, you'll skip the prog_array with the risk of missing
121     * out of events when it was updated in between this and the
122     * rcu_dereference() which is accepted risk.
123     */
124    ret = BPF_PROG_RUN_ARRAY_CHECK(call->prog_array, ctx, BPF_PROG_RUN);
125
126 out:
127    __this_cpu_dec(bpf_prog_active);
128
129    return ret;
130 }
131
132 #ifdef CONFIG_BPF_KPROBE_OVERRIDE
133 BPF_CALL_2(bpf_override_return, stru
134 {
135     regs_set_return_value(regs, rc);
136     override_function_with_return(re
137     return 0;
```

Extensions have been modified on disk. Please reload the window.

Reload Window

Nano Code 欢迎您-gdman 2.7.31

系统调用 → 修补内核 → 触发回调 → 输出事件 → 调试方法

```
int hello(void *ctx){
    bpf_trace_printk("Here is YourLand!\\n");
    return 0;
}
```

```
kn
 # Frame Base         Return Address    Call Site
00 ffffffc0`1803bb20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
01 ffffffc0`1803bbb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
02 ffffffc0`1803bbe0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
03 ffffffc0`1803bc30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8
[arch/arm64/kernel/probes/kprobes.c @ 353]
04 ffffffc0`1803bc40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-monitors.c @
326]
05 ffffffc0`1803bc60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
06 ffffffc0`1803bca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
07 ffffffc0`1803bcd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
08 ffffffc0`1803bce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @
223]
09 ffffffc0`1803be20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
0a ffffffc0`1803be40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0b ffffffc0`1803bcf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
0c ffffffc0`1803bec0 0000007f`f6408b70 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
```

File Edit Selection View Go Run Terminal Help — Nano调试 - Nano Code

Settings | Nano主页 | Nano调试 ×

文件 视图 输出 调试 高级 帮助

```
lk!trace_call_bpf:
ffffffc0101facf0 d503233f paciasp
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int

k
Frame Base        Return Address    Call Site
ffffffc0`17733b20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
ffffffc0`17733bb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
ffffffc0`17733be0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
ffffffc0`17733c30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8 [arch/arm64/kernel/probes/kprobes.c @ 353]
ffffffc0`17733c40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-monitors.c @ 326]
ffffffc0`17733c60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
ffffffc0`17733ca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
ffffffc0`17733cd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
ffffffc0`17733ce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @ 223]
ffffffc0`17733e20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
ffffffc0`17733e40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
ffffffc0`17733cf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
ffffffc0`17733ec0 00000000`00000000 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int

kn
 # Frame Base        Return Address    Call Site
00 ffffffc0`17733b20 ffffffc0`101fcc0c lk!trace_call_bpf [kernel/trace/bpf_trace.c @ 93]
01 ffffffc0`17733bb0 ffffffc0`101fe404 lk!kprobe_perf_func+0x5c [kernel/trace/trace_kprobe.c @ 1592]
02 ffffffc0`17733be0 ffffffc0`11563d68 lk!kprobe_dispatcher+0x80 [kernel/trace/trace_kprobe.c @ 1738]
03 ffffffc0`17733c30 ffffffc0`10016cb0 lk!kprobe_breakpoint_handler+0xc8 [arch/arm64/kernel/probes/kprobes.c @ 353]
04 ffffffc0`17733c40 ffffffc0`10017048 lk!call_break_hook+0x70 [arch/arm64/kernel/debug-monitors.c @ 326]
05 ffffffc0`17733c60 ffffffc0`1003aa14 lk!brk_handler+0x28 [arch/arm64/kernel/debug-monitors.c @ 332]
06 ffffffc0`17733ca0 ffffffc0`115558c4 lk!do_debug_exception+0xc4 [arch/arm64/mm/fault.c @ 968]
07 ffffffc0`17733cd0 ffffffc0`11556100 lk!el1_dbg+0x34 [arch/arm64/kernel/entry-common.c @ 186]
08 ffffffc0`17733ce0 ffffffc0`10012408 lk!el1_sync_handler+0xa0 [arch/arm64/kernel/entry-common.c @ 223]
09 ffffffc0`17733e20 ffffffc0`10095900 lk!el1_sync+0x88 [arch/arm64/kernel/entry.S @ 771]
0a ffffffc0`17733e40 ffffffc0`1002b8ec lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0b ffffffc0`17733cf0 ffffffc0`10095900 lk!do_el0_svc+0x7c [arch/arm64/kernel/syscall.c @ 195]
0c ffffffc0`17733ec0 00000000`00000000 lk!__arm64_sys_clone [kernel/fork.c @ 2643]
0xffffffc0`101facf4 struct trace_event_call *trace_event_call*0x00000000`d503233f*<Memory access error>unsigned int
```

6: kd>

entry.S | process.c | bpf_trace.c × | 反汇编.nd

```
86  * Return: BPF programs always return an integer which is interpreted by
87  * kprobe handler as:
88  * 0 - return from kprobe (event is filtered out)
89  * 1 - store kprobe event into ring buffer
90  * Other values are reserved and currently alias to 1
91  */
92 unsigned int trace_call_bpf(struct trace_event_call *call, void *ctx)
93 {
94     unsigned int ret;
95
96     cant_sleep();
97
98     if (unlikely(__this_cpu_inc_return(bpf_prog_active) != 1)) {
99         /*
100         * since some bpf program is already running on this cpu,
101         * don't call into another bpf program (same or different)
102         * and don't send kprobe event into ring-buffer,
103         * so return zero here
104         */
105        ret = 0;
106        goto out;
107    }
108
109    /*
110     * Instead of moving rcu_read_lock/rcu_dereference/rcu_read_unlock
111     * to all call sites, we did a bpf_prog_array_valid() there to check
112     * whether call->prog_array is empty or not, which is
113     * a heurisitc to speed up execution.
114     *
115     * If bpf_prog_array_valid() fetched prog_array was
116     * non-NULL, we go into trace_call_bpf() and do the actual
117     * proper rcu_dereference() under RCU lock.
118     * If it turns out that prog_array is NULL then, we bail out.
119     * For the opposite, if the bpf_prog_array_valid() fetched pointer
120     * was NULL, you'll skip the prog_array with the risk of missing
121     * out of events when it was updated in between this and the
122     * rcu_dereference() which is accepted risk.
123     */
124    ret = BPF_PROG_RUN_ARRAY_CHECK(call->prog_array, ctx, BPF_PROG_RUN);
125
126 out:
127    __this_cpu_dec(bpf_prog_active);
128
129    return ret;
130 }
131
132 #ifdef CONFIG_BPF_KPROBE_OVERRIDE
133 BPF_CALL_2(bpf_override_return, stru
134 {
135     regs_set_return_value(regs, rc);
136     override_function_with_return(re
137     return 0;
```

Extensions have been modified on disk. Please reload the window.

Reload Window

Nano Code 欢迎您-gdman 2.7.31

```
dt call
Local var @ x0 Type trace_event_call*
   +0x000 list             : list_head
   +0x010 class            : 0xffffff81`a1882408
   +0x018                  :
   +0x020 event            : trace_event
   +0x050 print_fmt        : 0xffffff81`946f9280  ""(%lx)", REC->__probe_ip"
   +0x058 filter           : (null)
   +0x060 mod              : (null)
   +0x068 data             : (null)
   +0x070 flags            : 32
   +0x074 perf_refcount    : 1
   +0x078 perf_events      : 0x00000039`d538a210
   +0x080 prog_array       : 0xffffff81`946f9c00
   +0x088 perf_perm        : (null)
```

```
r
 x0=0xffffffc01721ba58  x1=0x0000000000000013  x2=0xffffffc00013110c  x3=0x0000000000000002
 x4=0x0000000000000000  x5=0x0000000000000000  x6=0xffffffc0101fad24  x7=0x0000000000000000
 x8=0x000000000000007f  x9=0x0000000000000001 x10=0xffffffc0101f8d84 x11=0x0000000000000000
x12=0x0000000000000000 x13=0x0000000000000000 x14=0x0000000000000000 x15=0x0000000000000000
x16=0x0000000000000000 x17=0x0000000000000000 x18=0x0000000000000000 x19=0xffffffc01477f000
x20=0xffffffc01721bce0 x21=0x0000000000000001 x22=0xffffffc012c4b450 x23=0xffffffc01721bce0
x24=0xffffffc013b23490 x25=0xffffffc01721ba70 x26=0x0000000000000000 x27=0x0000000000000000
x28=0xffffff8138d69600  fp=0xffffffc01721baa0  lr=0xffffffc000131198  sp=0xffffffc01721ba50
 pc=0xffffffc0101f8d84 pstate=0x00000000604003c9 - Z C - - - - P - D A I F EL1
```

```
db 0xffffffc01721ba58
ffffffc0`1721ba58  48 65 72 65 20 69 73 20-59 6f 75 72 4c 61 6e 64  Here is YourLand
ffffffc0`1721ba68  21 0a 00 12 c0 ff ff ff-c8 b7 39 13 c0 ff ff ff  !.........9.....
ffffffc0`1721ba78  10 74 aa 20 81 ff ff ff-01 00 00 00 00 00 00 00  .t. ............
ffffffc0`1721ba88  50 b4 c4 12 c0 ff ff ff-00 f0 77 14 c0 ff ff ff  P.........w.....
ffffffc0`1721ba98  e0 bc 21 17 c0 ff ff ff-b0 ba 21 17 c0 ff ff ff  ..!.......!.....
ffffffc0`1721baa8  9c ad 1f 10 c0 ff ff ff-20 bb 21 17 c0 ff ff ff  ........ .!.....
ffffffc0`1721bab8  0c cc 1f 10 c0 ff ff ff-00 59 09 10 c0 ff ff ff  .........Y......
ffffffc0`1721bac8  e0 bc 21 17 c0 ff ff ff-e0 bc 21 17 c0 ff ff ff  ..!.......!.....
```

```
root@ulan:~# ls -l /proc/2739/fd
total 0
lrwx------ 1 root root 64 Oct 16 17:13 0 -> /dev/pts/2
lrwx------ 1 root root 64 Oct 16 17:31 1 -> /dev/pts/2
lrwx------ 1 root root 64 Oct 16 17:31 2 -> /dev/pts/2
lr-x------ 1 root root 64 Oct 16 17:11 3 -> anon_inode:btf
lrwx------ 1 root root 64 Oct 16 17:31 4 -> anon_inode:bpf-prog
lrwx------ 1 root root 64 Oct 16 17:31 5 -> 'anon_inode:[perf_event]'
lr-x------ 1 root root 64 Oct 16 17:31 6 -> /sys/kernel/debug/tracing/trace_pipe
```

系统调用 → 修补内核 → 触发回调 → 输出事件 → 调试方法

调试工具1：strace

- 监视系统调用
- 从头监视
  - strace <命令行>
- 中途附加
  - strace -p 2739

监视收到的事件数据

```
root@ulan:~# strace -e read=6 -e trace=read -p 2739
strace: Process 2739 attached
read(6, "             bash-3109    [000] d"..., 4096) = 87
  | 00000  20 20 20 20 20 20 20 20  20 20 20 20 62 61 73 68                bash |
  | 00010  2d 33 31 30 39 20 20 20  20 5b 30 30 30 5d 20 64  -3109    [000] d |
  | 00020  2e 2e 2e 20 20 31 33 35  30 2e 30 34 33 34 39 33  ...  1350.043493 |
  | 00030  3a 20 62 70 66 5f 74 72  61 63 65 5f 70 72 69 6e  : bpf_trace_prin |
  | 00040  74 6b 3a 20 48 65 72 65  20 69 73 20 59 6f 75 72  tk: Here is Your |
  | 00050  4c 61 6e 64 21 0a 0a                              Land!..          |
read(6,
```

调试工具1：bpftool

- 集成在内核代码主线里的bpf工具
- tools/bpf/bpftool
- tools/lib/bpf
- 也可以使用git上的项目仓库
  - https://github.com/libbpf/bpftool

```
geduer@ulan:/gewu/bpftool/src$ bpftool
Usage: bpftool [OPTIONS] OBJECT { COMMAND | help }
       bpftool batch file FILE
       bpftool version

       OBJECT := { prog | map | link | cgroup | perf | net | feature | btf | gen | struct_ops | iter }
       OPTIONS := { {-j|--json} [{-p|--pretty}] | {-d|--debug} |
                    {-V|--version} }
```

```
geduer@ulan:/gewu/ulan-5.10-rk3588/tools/bpf/bpftool$ make

make[1]: Entering directory '/gewu/ulan-5.10-rk3588/tools/lib/bpf'

Auto-detecting system features:
...                        libelf: [ on  ]
...                          zlib: [ on  ]
...                           bpf: [ on  ]

  GEN      bpf_helper_defs.h
  MKDIR    staticobjs/
  CC       staticobjs/libbpf.o
  CC       staticobjs/bpf.o
  CC       staticobjs/nlattr.o
  CC       staticobjs/btf.o
  CC       staticobjs/libbpf_errno.o
  CC       staticobjs/str_error.o
  CC       staticobjs/netlink.o
  CC       staticobjs/bpf_prog_linfo.o
  CC       staticobjs/libbpf_probes.o
  CC       staticobjs/xsk.o
  CC       staticobjs/hashmap.o
  CC       staticobjs/btf_dump.o
  CC       staticobjs/ringbuf.o
  LD       staticobjs/libbpf-in.o
  LINK     libbpf.a
make[1]: Leaving directory '/gewu/ulan-5.10-rk3588/tools/lib/bpf'
  LINK     bpftool
```

```
geduer@ulan:/gewu/ulan-5.10-rk3588/tools/bpf/bpftool$ sudo ./bpftool prog list
3: cgroup_device  name sd_devices  tag a42d275341448247  gpl
        loaded_at 2024-10-15T09:56:18+0800  uid 0
        xlated 504B  jited 472B  memlock 4096B
4: cgroup_skb  name sd_fw_egress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-10-15T09:56:18+0800  uid 0
        xlated 64B  jited 104B  memlock 4096B
5: cgroup_skb  name sd_fw_ingress  tag 6deef7357e7b4530  gpl
        loaded_at 2024-10-15T09:56:18+0800  uid 0
        xlated 64B  jited 104B  memlock 4096B
```

```
44: kprobe  name hello  tag 692e7f7aa8f18bf6  gpl
        loaded_at 2024-10-15T10:48:57+0800  uid 0
        xlated 128B  jited 176B  memlock 4096B
        btf_id 1
```

调试工具3：Linux内核调试

- 虚拟机环境
  - KDB/KGDB
  - QEMU
- 真机环境
  - 挥码枪硬件调试器
  - 主机端可以是Windows或者Linux

格物致知

https://nanocode.cn/