Debugging for Linux and Android

Peter Liu
Sept 2021

Peter Liu, Oct 2021

CONFIDENTIAL AND PROPRIETARY

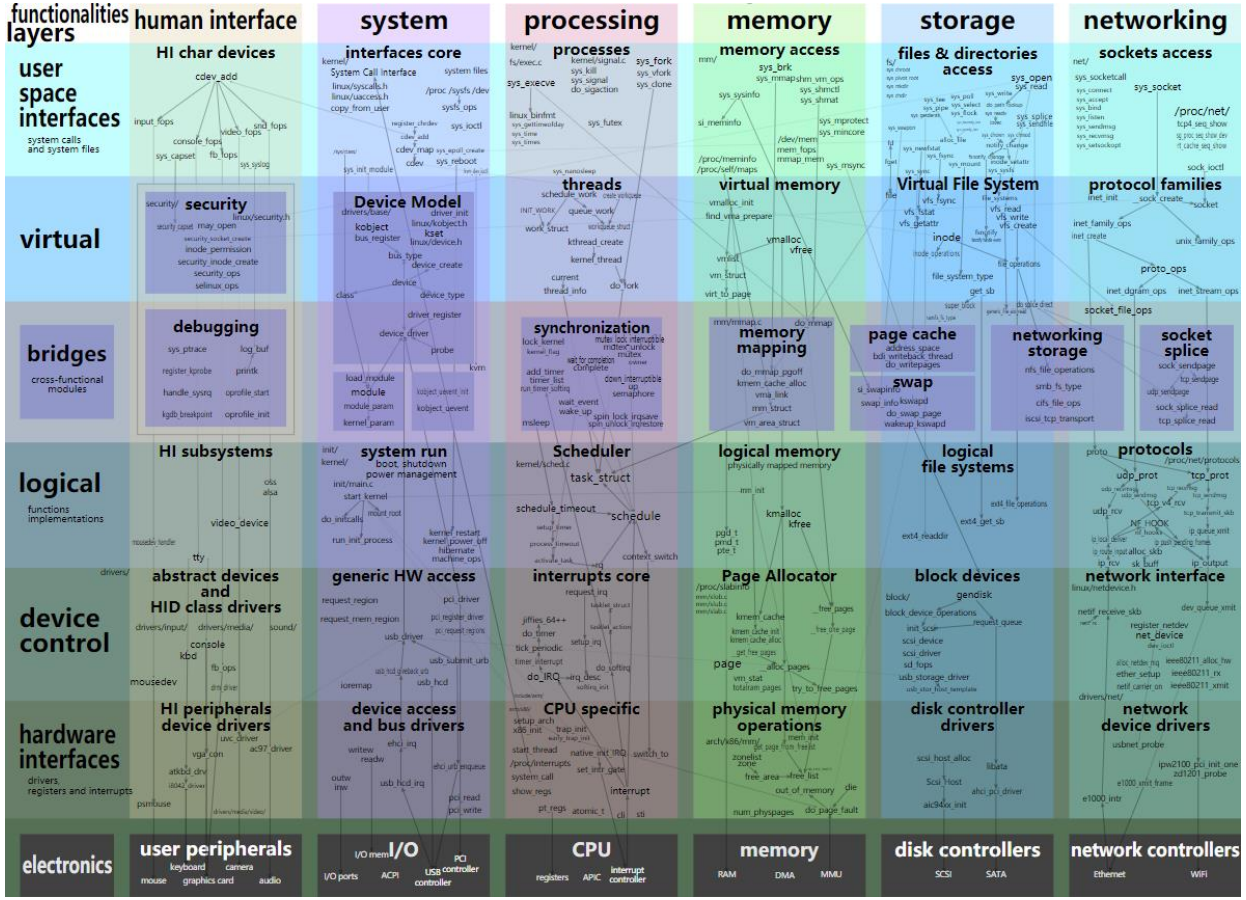
Agenda

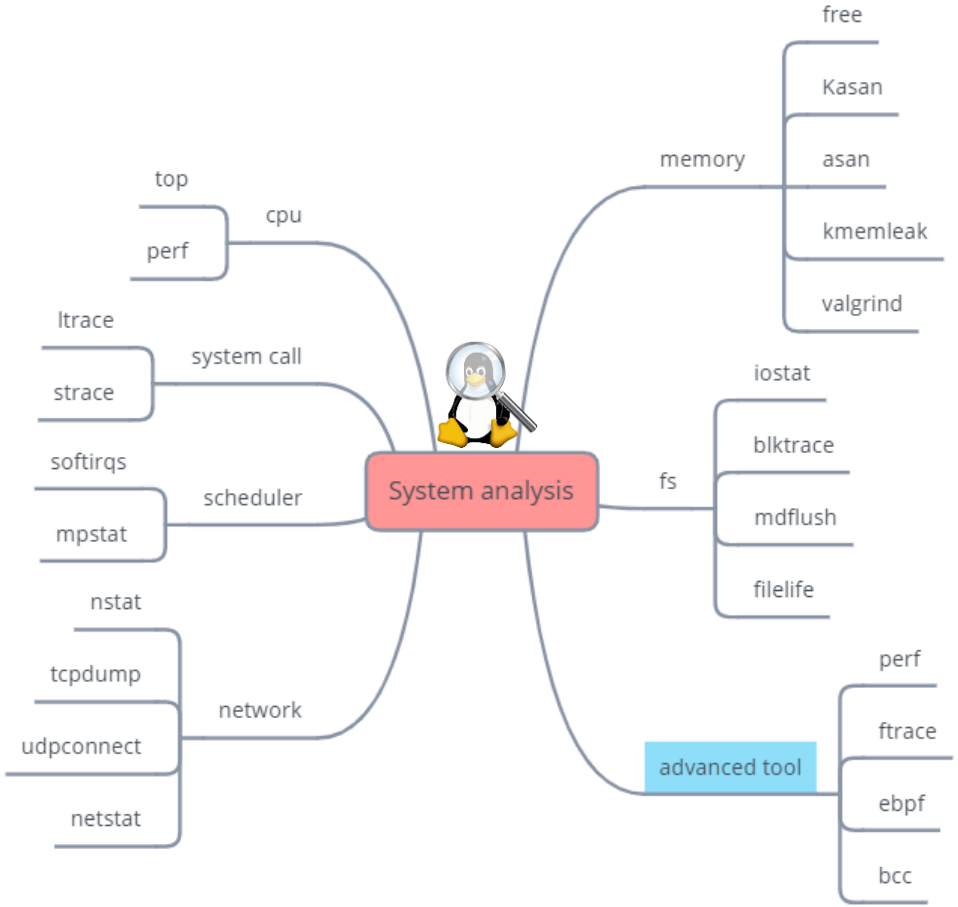
- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

OS and System analysis

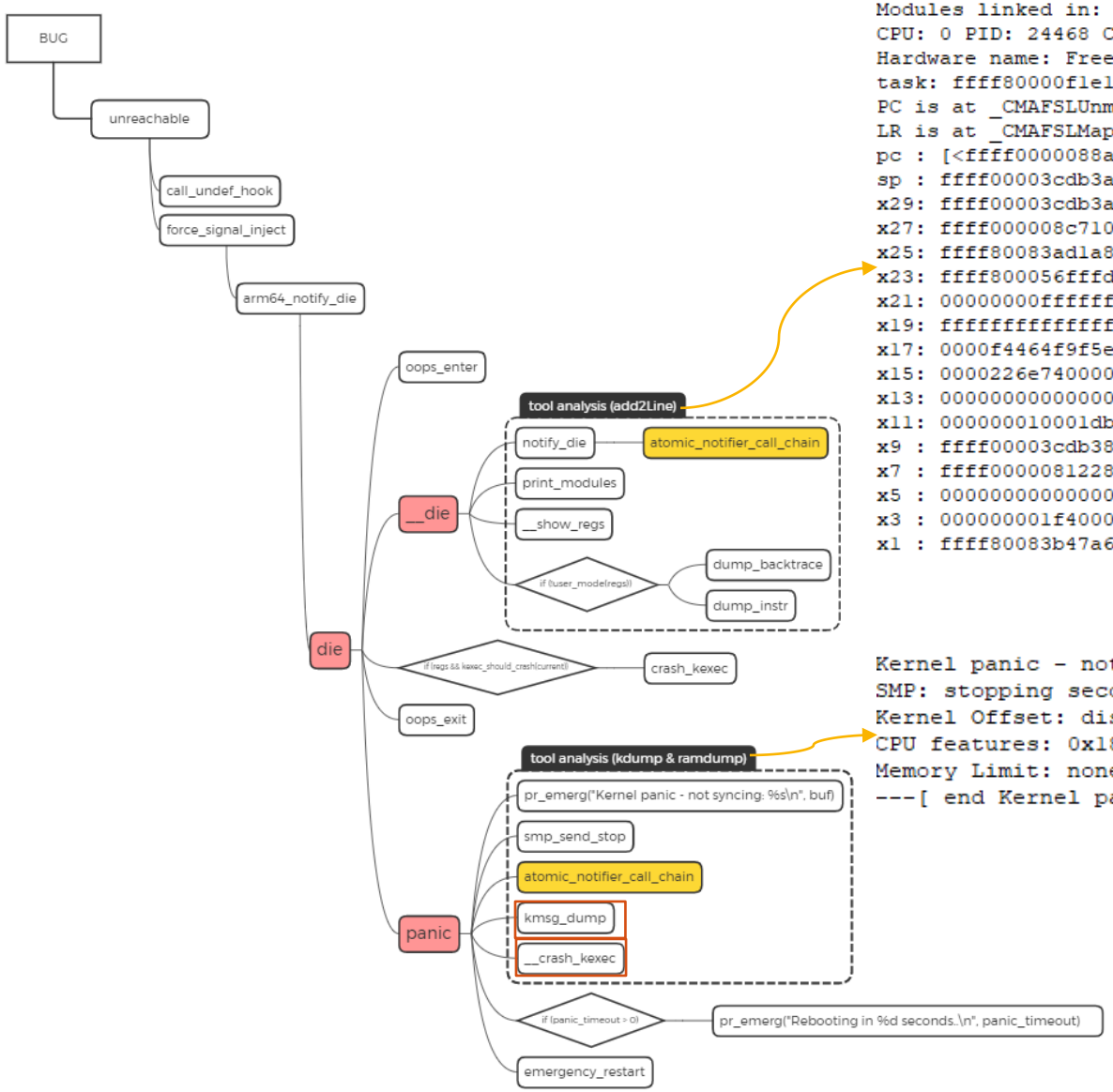




Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

Oops/Panic case




```
Internal error: Oops: 96000006 [#1] PREEMPT SMP
Modules linked in: cyttsp6_i2c cyttsp6_device_access cyttsp6_loader c
CPU: 0 PID: 24468 Comm: EGL Init Tainted: G        W  O    4.14.98-0
Hardware name: Freescale i.MX8QM MEK (DT)
task: ffff80000fle1580 task.stack: ffff00003cdb0000
PC is at _CMAFSLUnmapUser+0x14/0x28
LR is at _CMAFSLMapUser+0x108/0x124
pc : [<ffff0000088a8b1c>] lr : [<ffff0000088a8d10>] pstate: 60000145
sp : ffff00003cdb3a50
x29: ffff00003cdb3a50 x28: ffff80000fle1580
x27: ffff000008c71000 x26: 0000000000000000
x25: ffff80083ad1a800 x24: 0000000000000000
x23: ffff800056fffd00 x22: ffff80083b47a638
x21: 00000000fffffffd x20: ffff80083b47a600
x19: fffffffffffffffc x18: 0000f44616327268
x17: 0000f4464f9f5e74 x16: ffff00000827f960
x15: 0000226e74000000 x14: 0000000000000000
x13: 0000000000000000 x12: 0000000000000000
x11: 000000010001dbf3 x10: 00000000000011a0
x9 : ffff00003cdb3830 x8 : 287265735570614d
x7 : ffff0000081228ac x6 : 0000000000000000
x5 : 0000000000000000 x4 : 0000000000000000
x3 : 000000001f400000 x2 : fffffffffffffffc
x1 : ffff80083b47a600 x0 : ffff800834b93a80
```

```
Kernel panic - not syncing: Fatal exception
SMP: stopping secondary CPUs
Kernel Offset: disabled
CPU features: 0x180200d
Memory Limit: none
---[ end Kernel panic - not syncing: Fatal exception
```


CONFIDENTIAL AND PROPRIETARY

Oops/Panic case

- addr2line

```
./aarch64-linux-android-addr2line -e vmlinux 0xffff0000087b2484
/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:350
```

- objdump

```
00000000000002a8 <_CMAFSLUnmapUser>:
_CMAFSLUnmapUser():
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:342
{
 2a8:   a9bf7bfd        stp     x29, x30, [sp,#-16]!
get_current():
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/arch/arm64/include/asm/current.h:19
 */
static __always_inline struct task_struct *get_current(void)
{
        unsigned long sp_el0;

        asm ("mrs %0, sp_el0" : "=r" (sp_el0));
 2ac:   d5384100        mrs     x0, sp_el0
_CMAFSLUnmapUser():
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:342
 2b0:   910003fd        mov     x29, sp
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:343
    if (unlikely(current->mm == gcvNULL))
 2b4:   f941ec00        ldr     x0, [x0,#984]
 2b8:   b4000080        cbz     x0, 2c8 <_CMAFSLUnmapUser+0x20>
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:350
    if (vm_munmap((unsigned long)MdlMap->vmaAddr, Size) < 0)
 2bc:   f9400840        ldr     x0, [x2,#16]
 2c0:   2a0303e1        mov     w1, w3
 2c4:   94000000        bl      0 <vm_munmap>
/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:370
}
 2c8:   a8c17bfd        ldp     x29, x30, [sp],#16
 2cc:   d65f03c0        ret
```

- gdb

```
(gdb) b *_CMAFSLUnmapUser+0x14
Breakpoint 1 at 0x87b2484
(gdb) l *0xFFFF0000087B2484
0x87b2484 is in _CMAFSLUnmapUser (/opt/samba/nxf39444/p9.0.0_2.3.0/vendor/nxp-opensource/kernel_imx/drivers/mxc/gpu-viv/hal/os/linux/kernel/allocator/freescale/gc_hal_kernel_allocator_cma.c:350).
345             /* Do nothing if process is exiting. */
346             return;
347         }
348
349     #if LINUX_VERSION_CODE >= KERNEL_VERSION(3,4,0)
350         if (vm_munmap((unsigned long)MdlMap->vmaAddr, Size) < 0)
351         {
352             gcmkTRACE_ZONE(
353                     gcvLEVEL_WARNING, gcvZONE_OS,
354                     "%s(%d): vm_munmap failed",
```


CONFIDENTIAL AND PROPRIETARY

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

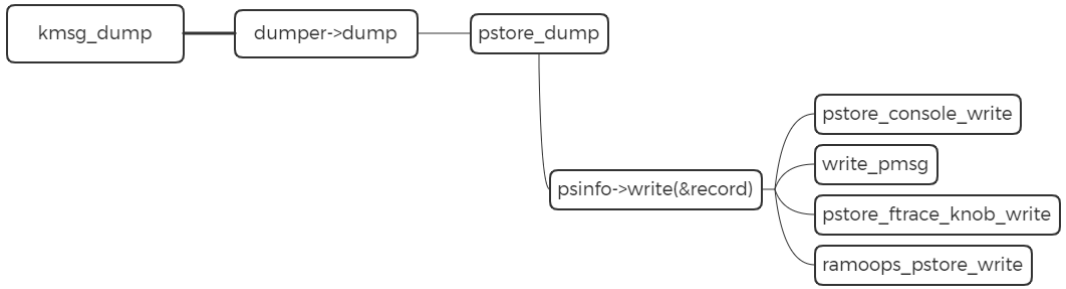
Pstore

Pstore is a filesystem which provides a generic interface to capture kernel records in the dying moments or we could redefine it as a generic interface to capture kernel records that will persist across reboots.

Pstore supports different types of records. Some of the commonly used:

- PSTORE_CONSOLE
  - Log kernel console messages
- PSTORE_PMSG
  - Log user space messages
- PSTORE_FTRACE
  - Persistent function tracer
- PSTORE_RAM
  - Log oops/panic to a RAM buffer

```
ramoops@0x91f00000 {
        compatible = "ramoops";
        reg = <0 0x91f00000 0 0x00100000>;
        record-size     = <0x00020000>;
        console-size    = <0x00020000>;
        ftrace-size     = <0x00020000>;
        pmsg-size       = <0x00020000>;
};
```



CONFIDENTIAL AND PROPRIETARY

Pstore Ramoops backend

Ramoops is a backend interface which enables pstore records to use persistent RAM as their storage to survive across reboots.

Records are stored in following format in pstore filesystem and can be read after mounting pstore:

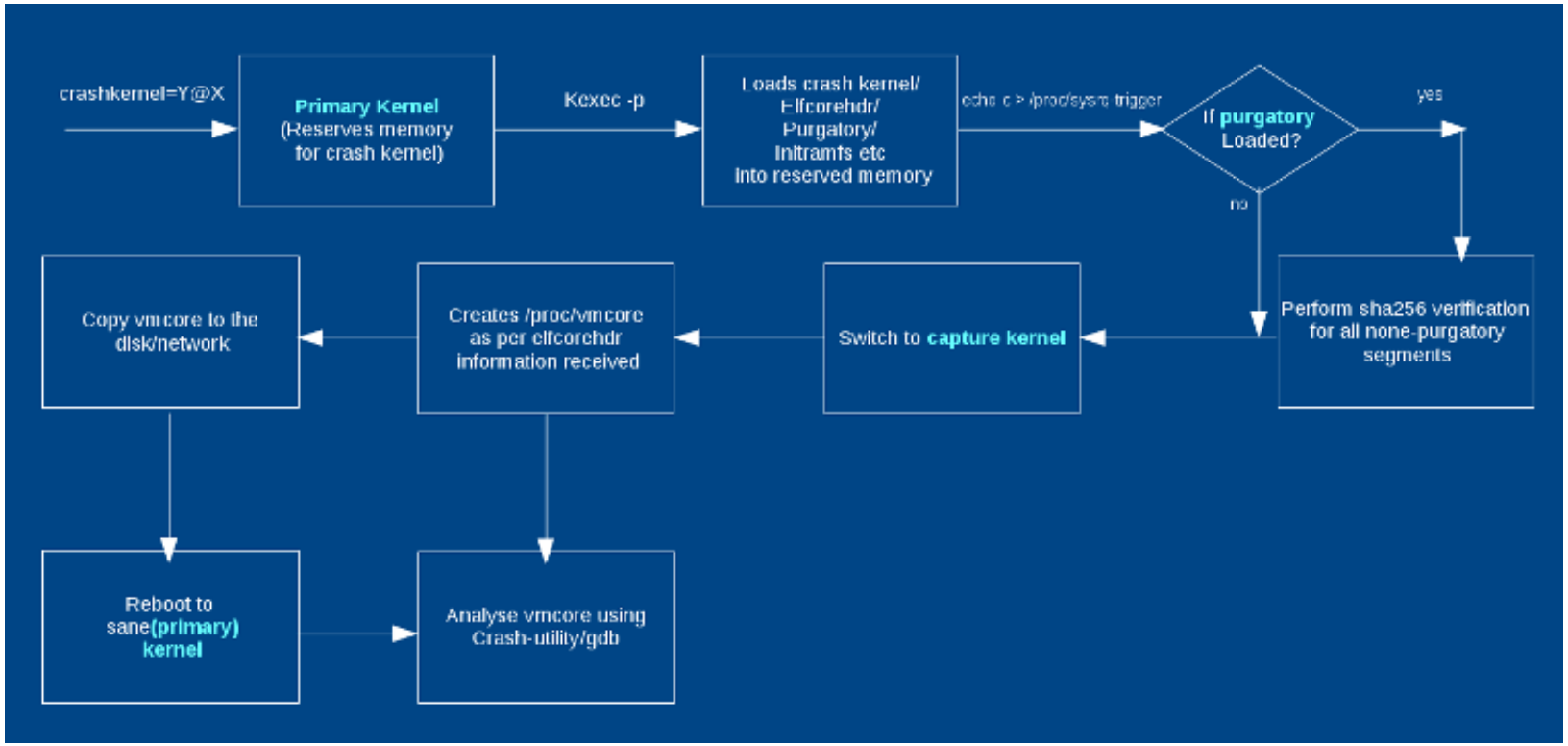
- console-ramoops
- dmesg-ramoops
- ftrace-ramoops
- pmsg-ramoops

```
# mount -t debugfs debugfs /sys/kernel/debug/
# echo 1 > /sys/kernel/debug/pstore/record_ftrace
# reboot -f
[...]
# mount -t pstore pstore /mnt/
# tail /mnt/ftrace-ramoops
0 ffffffff8101ea64  ffffffff8101bcda  native_apic_mem_read <- disconnect_bsp_APIC+0x6a/0xc0
0 ffffffff8101ea44  ffffffff8101bcf6  native_apic_mem_write <- disconnect_bsp_APIC+0x86/0xc0
0 ffffffff81020084  ffffffff8101a4b5  hpet_disable <- native_machine_shutdown+0x75/0x90
0 ffffffff81005f94  ffffffff8101a4bb  iommu_shutdown_noop <- native_machine_shutdown+0x7b/0x90
0 ffffffff8101a6a1  ffffffff8101a437  native_machine_emergency_restart <- native_machine_restart+0x37/0x40
0 ffffffff811f9876  ffffffff8101a73a  acpi_reboot <- native_machine_emergency_restart+0xaa/0x1e0
0 ffffffff8101a514  ffffffff8101a772  mach_reboot_fixups <- native_machine_emergency_restart+0xe2/0x1e0
0 ffffffff811d9c54  ffffffff8101a7a0  __const_udelay <- native_machine_emergency_restart+0x110/0x1e0
0 ffffffff811d9c34  ffffffff811d9c80  __delay <- __const_udelay+0x30/0x40
0 ffffffff811d9d14  ffffffff811d9c3f  delay_tsc <- __delay+0xf/0x20
```

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

Kdump



- Switch to capture kernel when system crash by **kexec** command.
- Using **crash** to debug /proc/vmcore on target.

Kdump

- $ crash vmlinux crash.dump

```
crash> help

*               extend          log             rd              task
alias           files           mach            repeat          timer
ascii           foreach         mod             runq            tree
bpf             fuser           mount           search          union
bt              gdb             net             set             vm
btop            help            p               sig             vtop
dev             ipcs            ps              struct          waitq
dis             irq             pte             swap            whatis
eval            kmem            ptob            sym             wr
exit            list            ptov            sys             q

crash version: 7.2.8    gdb version: 7.6
For help on any command above, enter "help <command>".
For help on input options, enter "help input".
For help on output options, enter "help output".
```

- View the Process when System Crashed
- View Swap space when System Crashed
- View IRQ when System Crashed
- View the Virtual Memory when System Crashed
- View System Information when System Crashed
- View the virtual memory usage when the system crashed

Kdump

```
crash> bt
PID: 4078   TASK: ffff8008343dc380  CPU: 1   COMMAND: "sh"
 #0 [ffff00002a493860] machine_kexec at ffff0000080964b0
 #1 [ffff00002a4938b0] __crash_kexec at ffff000008152230
 #2 [ffff00002a493a20] crash_kexec at ffff000008152330
 #3 [ffff00002a493a50] die at ffff00000808a14c
 #4 [ffff00002a493a90] __do_kernel_fault at ffff00000809bfa0
 #5 [ffff00002a493ac0] do_page_fault at ffff00000809c0c4
 #6 [ffff00002a493b30] do_translation_fault at ffff00000809c488
 #7 [ffff00002a493b40] do_mem_abort at ffff00000808130c
 #8 [ffff00002a493d20] el1_ia at ffff000008083050
    PC: ffff000008604008  [sysrq_handle_crash+32]
    LR: ffff000008603ff4  [sysrq_handle_crash+12]
    SP: ffff00002a493d30  PSTATE: 60000145
   X29: ffff00002a493d30  X28: ffff8008343dc380  X27: ffff000008e11000
   X26: 0000000000000040  X25: 0000000000000124  X24: 0000000000000000
   X23: 0000000000000007  X22: ffff000009651000  X21: ffff000009651e50
   X20: 0000000000000063  X19: ffff000009582000  X18: 0000000000000010
   X17: 0000ffff984e2270  X16: ffff0000082165b0  X15: ffffffffffffffff
   X14: ffff000089736547  X13: ffff000009736555  X12: ffff000009568df8
   X11: ffff00000863bba0  X10: ffff00002a493a60   X9: 0000000000000007
    X8: 6767697254203a20   X7: 7152737953203a71   X6: 00000000000001cc
    X5: 0000000000000000   X4: 0000000000000000   X3: 0000000000000000
    X2: ffff80083fe35ef0   X1: 0000000000000000   X0: 0000000000000001
 #9 [ffff00002a493d30] sysrq_handle_crash at ffff000008604004
#10 [ffff00002a493d40] __handle_sysrq at ffff0000086045b4
#11 [ffff00002a493d80] write_sysrq_trigger at ffff000008604b58
#12 [ffff00002a493da0] proc_reg_write at ffff00000828212c
#13 [ffff00002a493dc0] __vfs_write at ffff00000821603c
#14 [ffff00002a493e40] vfs_write at ffff000008216330
#15 [ffff00002a493e80] sys_write at ffff0000082165f4
#16 [ffff00002a493ff0] __sys_trace at ffff000008083b14
    PC: 0000ffff98538d4c   LR: 0000ffff984e54dc   SP: 0000ffffca92fdf0
   X29: 0000ffffca92fdf0  X28: 000000000be3be70  X27: 0000000000000000
   X26: 0000000000000001  X25: 00000000004d21d8  X24: 0000000000000002
   X23: 000000000be3c610  X22: 0000000000000002  X21: 0000ffff985c7560
   X20: 000000000be3c610  X19: 0000000000000001  X18: 0000ffff985c6a70
   X17: 0000ffff984e2270  X16: 00000000004e93b8  X15: 0000000000000000
   X14: 0000000000000002  X13: 000000000000270f  X12: 0000000000000001
   X11: 0000000000000000  X10: 0000000000000010   X9: 0000ffff98635700
    X8: 0000000000000040   X7: 0000000000000001   X6: 0000000000000001
    X5: 5551000454000000   X4: 0000000000000000   X3: 0000ffff985cb190
    X2: 0000000000000002   X1: 000000000be3c610   X0: 0000000000000001
  ORIG_X0: 0000000000000001  SYSCALLNO: 40  PSTATE: 20000000
```

Process caused crash

Instructions caused crash

sp

fp

Parameter one in the func

Parameter two in the func

- Locate the address of PC and catch the stack.

```
crash> dis -l ffff000008604008
/home/nxf39444/data/8qxp_6layer/linux/linux-imx/drivers/tty/sysrq.c: 147
0xffff000008604008 <sysrq_handle_crash+32>:     strb    w0, [x1]
crash>
crash> dis -r ffff000008604008
0xffff000008603fe8 <sysrq_handle_crash>:        stp     x29, x30, [sp,#-16]!
0xffff000008603fec <sysrq_handle_crash+4>:      mov     x29, sp
0xffff000008603ff0 <sysrq_handle_crash+8>:      bl      0xffff000008127b68 <__rcu_read_unlock>
0xffff000008603ff4 <sysrq_handle_crash+12>:     adrp    x1, 0xffff000009730000 <_xen_start_info+832>
0xffff000008603ff8 <sysrq_handle_crash+16>:     mov     w0, #0x1                        // #1
0xffff000008603ffc <sysrq_handle_crash+20>:     str     w0, [x1,#504]
0xffff000008604000 <sysrq_handle_crash+24>:     dsb     st
0xffff000008604004 <sysrq_handle_crash+28>:     mov     x1, #0x0                        // #0
0xffff000008604008 <sysrq_handle_crash+32>:     strb    w0, [x1]
crash>
crash> dis -f ffff000008604008
0xffff000008604008 <sysrq_handle_crash+32>:     strb    w0, [x1]
0xffff00000860400c <sysrq_handle_crash+36>:     ldp     x29, x30, [sp],#16
0xffff000008604010 <sysrq_handle_crash+40>:     ret
crash>
```

```
static void sysrq_handle_crash(int key)
{
        char *killer = NULL;

        /* we need to release the RCU read lock here,
         * otherwise we get an annoying
         * 'BUG: sleeping function called from invalid context'
         * complaint from the kernel before the panic.
         */
        rcu_read_unlock();
        panic_on_oops = 1;              /* force panic */
        wmb();
        *killer = 1;

}
```

do_mem_abort

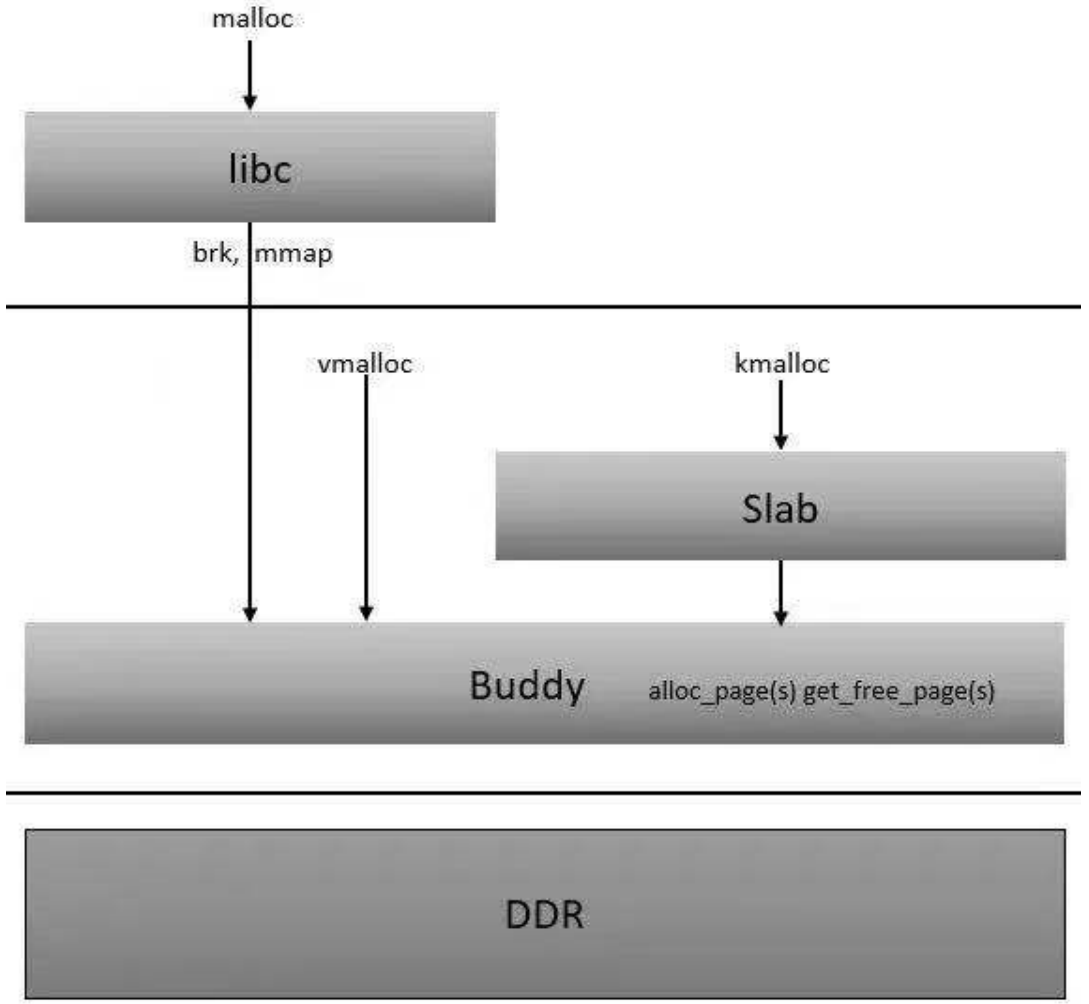
CONFIDENTIAL AND PROPRIETARY

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

Memory debugging

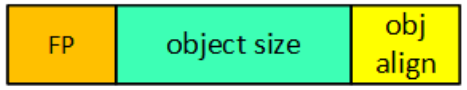
- out of bounds
- use after free
- use before initialize
- memory leak
- stack overflow



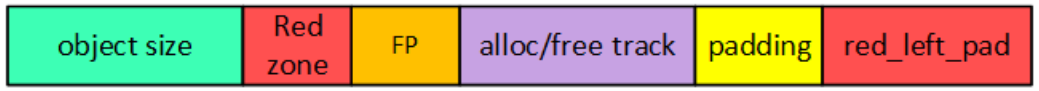
SLAB

- object layout

SLUB DEBUG is off



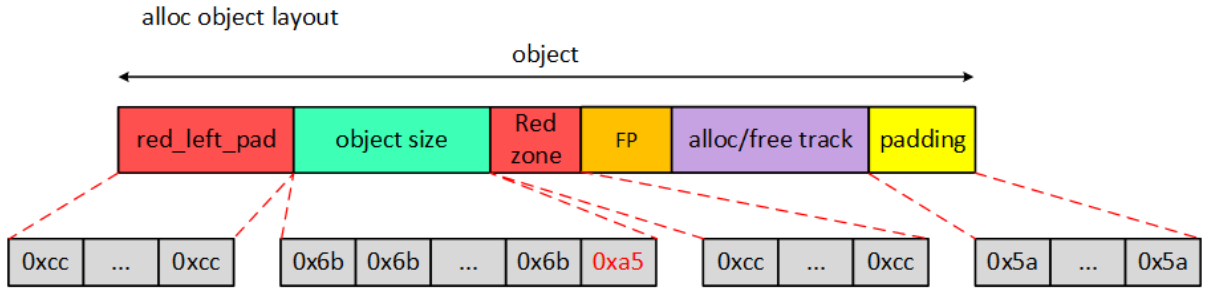
SLUB DEBUG is on with slub_debug=PZU



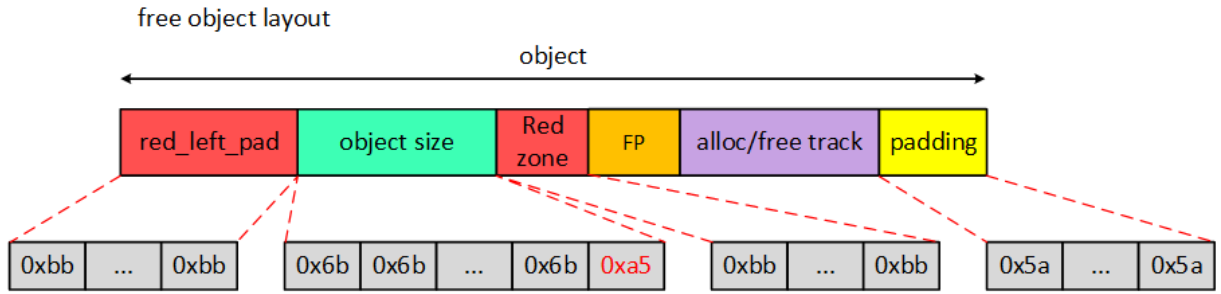
- magic num

```
//include/linux/poison.h
#define SLUB_RED_INACTIVE       0xbb
#define SLUB_RED_ACTIVE         0xcc

/* ...and for poisoning */
#define POISON_INUSE    0x5a    /* for use-uninitialised poisoning */
#define POISON_FREE     0x6b    /* for use-after-free poisoning */
#define POISON_END      0xa5    /* end-byte of poisoning */
```

- alloc object layout



- free object layout



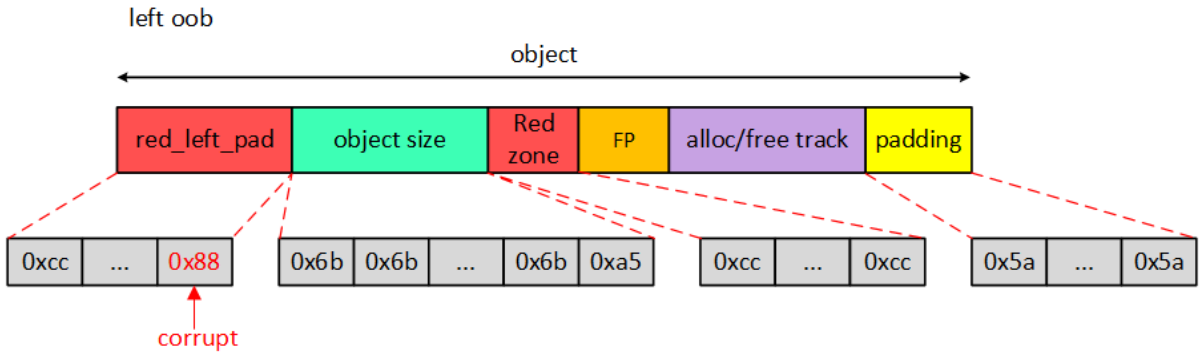
CONFIDENTIAL AND PROPRIETARY

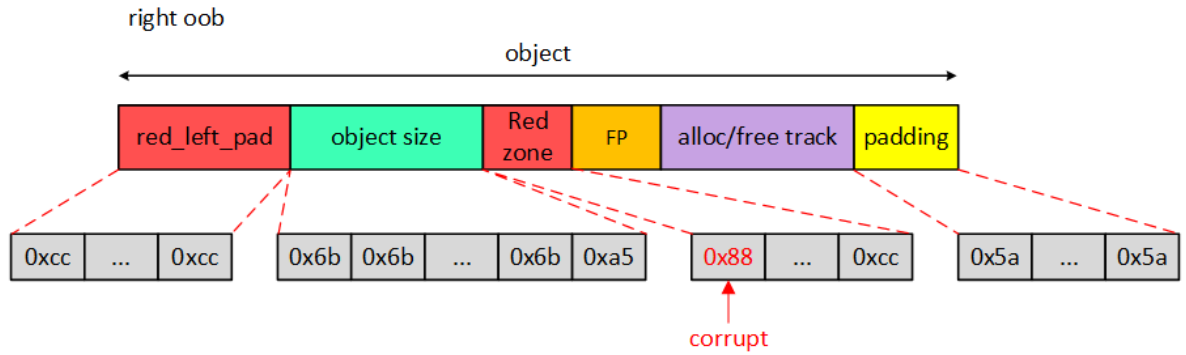
SLAB

- out of bounds

```
void left_oob(void)
{
    char *p = kmalloc(32, GFP_KERNEL);
    if (!p)
        return;
    p[-1] = 0x88;
    kfree(p);
}
```

```
void right_oob(void)
{
    char *p = kmalloc(32, GFP_KERNEL);
    if (!p)
        return;
    p[32] = 0x88;
    kfree(p);
}
```

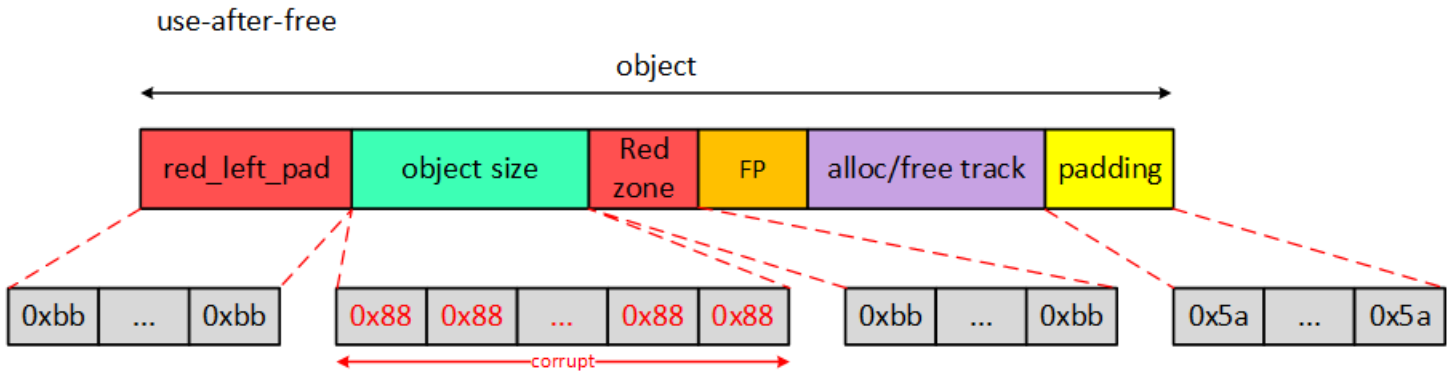




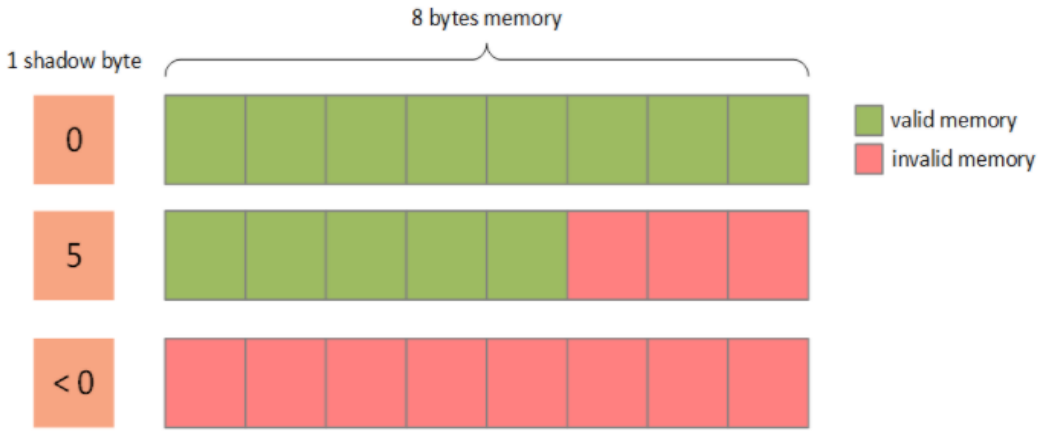
SLAB

- use after free

```
void use_after_free(void)
{
    char *p = kmalloc(32, GFP_KERNEL);
    if (!p)
        return;

    kfree(p);
    memset(p, 0x88, 32);
}
```



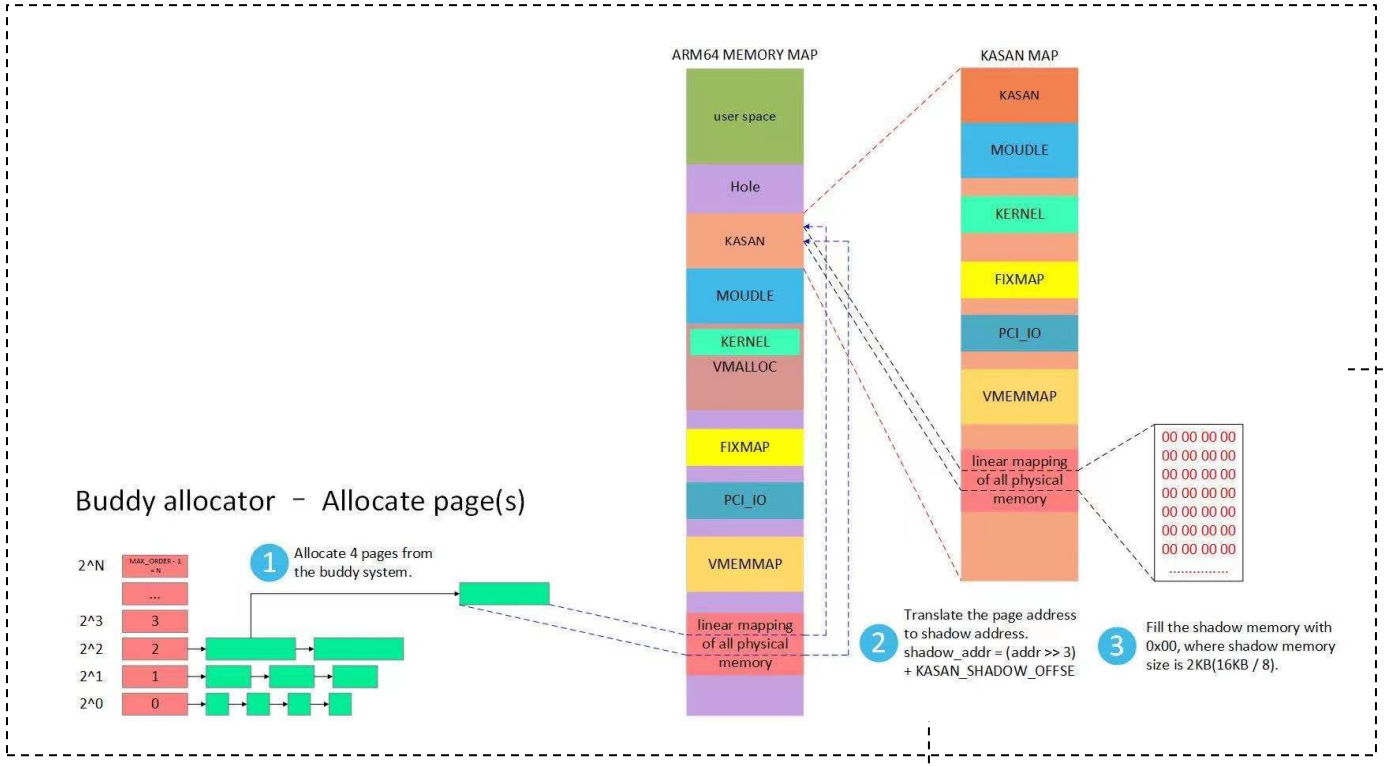
KASAN

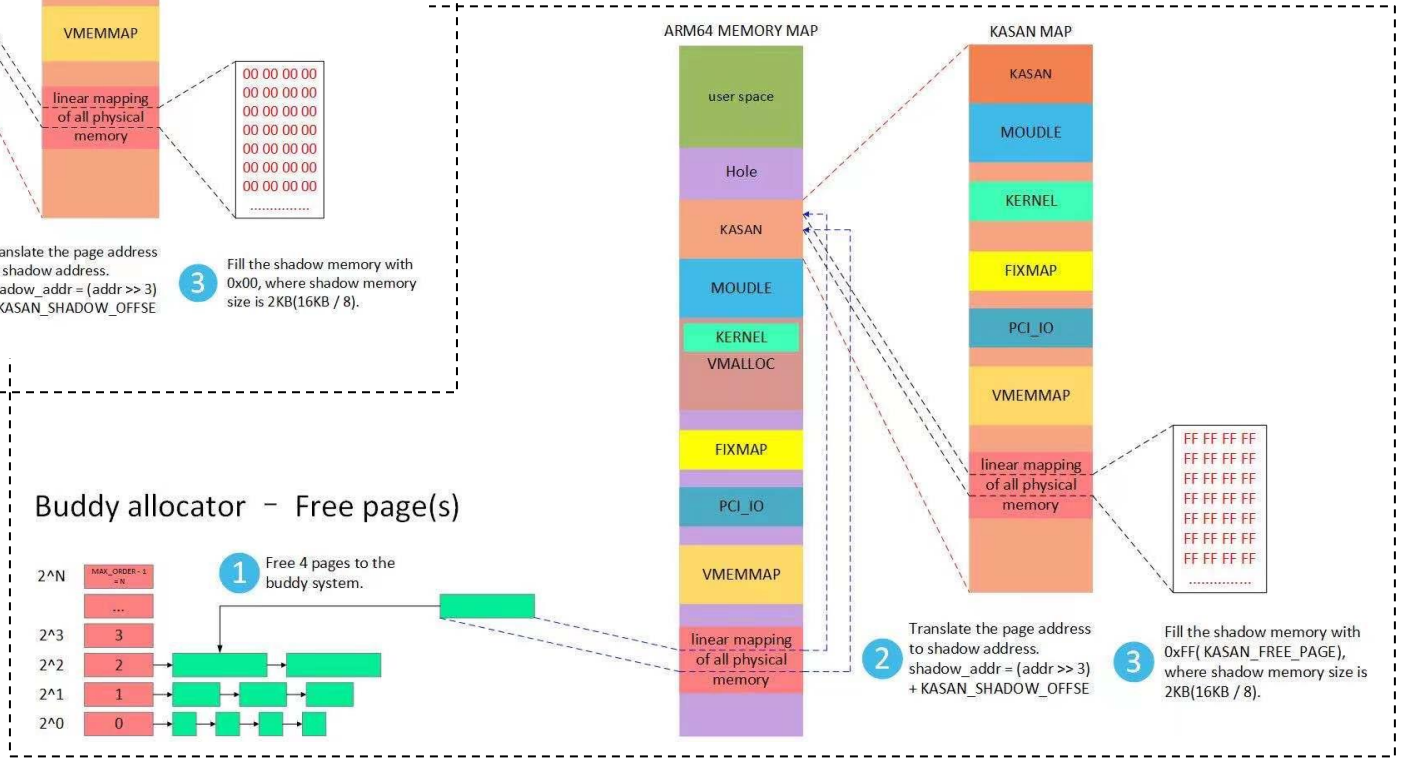



```
//mm/kasan/kasan.h
#define KASAN_FREE_PAGE         0xFF  /* page was freed */
#define KASAN_PAGE_REDZONE      0xFE  /* redzone for kmalloc_large allocations */
#define KASAN_KMALLOC_REDZONE   0xFC  /* redzone inside slub object */
#define KASAN_KMALLOC_FREE      0xFB  /* object was freed (kmem_cache_free/kfree) */
#define KASAN_GLOBAL_REDZONE    0xFA  /* redzone for global variable */
```


CONFIDENTIAL AND PROPRIETARY

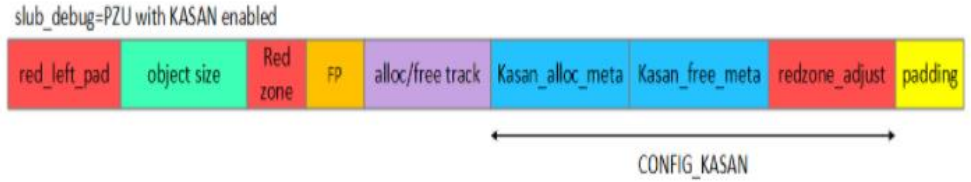
KASAN

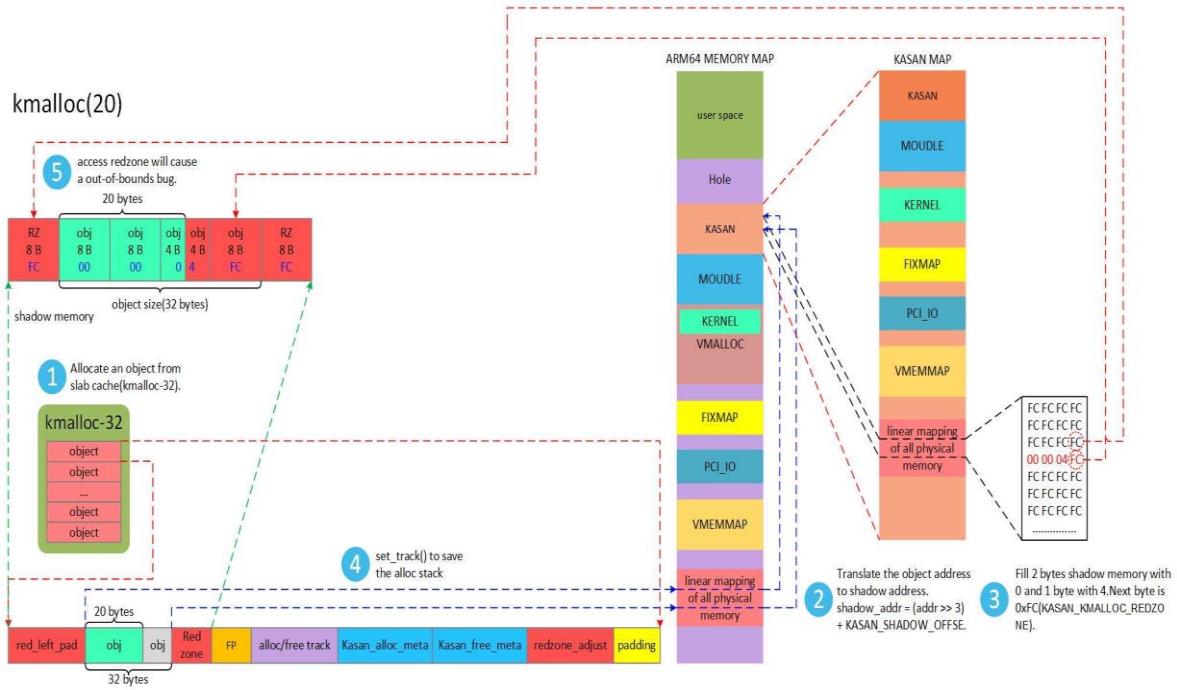


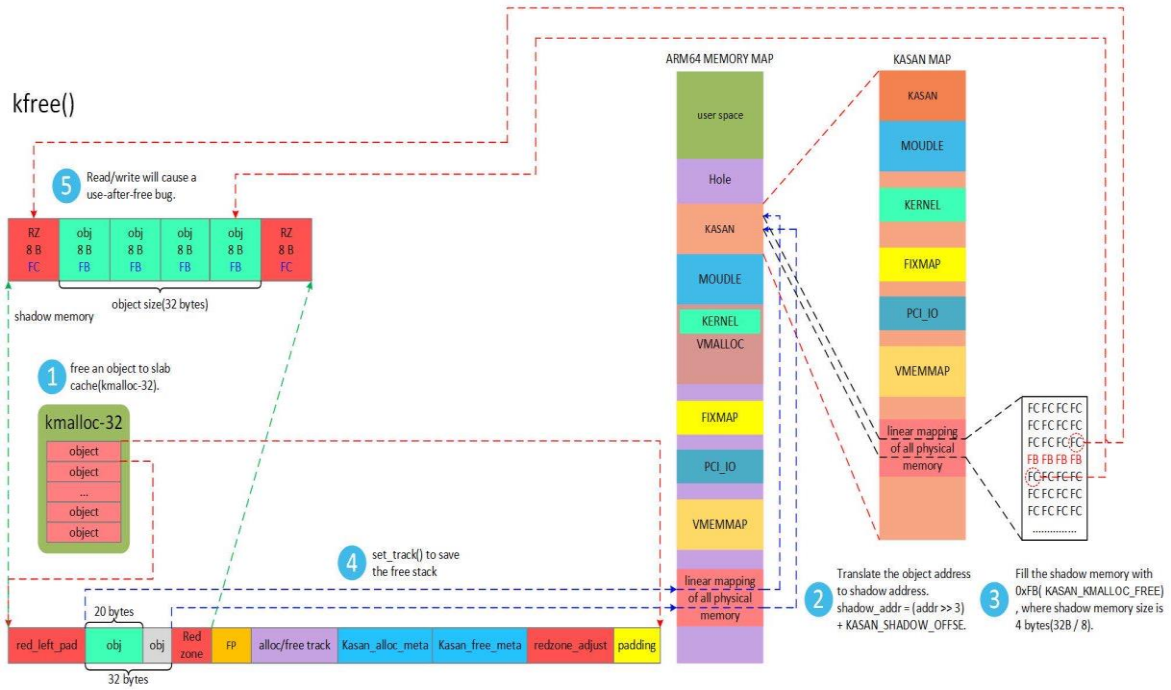


CONFIDENTIAL AND PROPRIETARY

KASAN


slub_debug=PZU with KASAN enabled
red_left_pad
object size
Red zone
FP
alloc/free track
Kasan_alloc_meta
Kasan_free_meta
redzone_adjust
padding
CONFIG_KASAN


kmalloc(20)
ARM64 MEMORY MAP
KASAN MAP
user space
Hole
KASAN
MOUDLE
KERNEL
VMALLOC
FIXMAP
PCI_IO
VMEMMAP
linear mapping of all physical memory
5 access redzone will cause a out-of-bounds bug.
20 bytes
object size(32 bytes)
shadow memory
1 Allocate an object from slab cache(kmalloc-32).
kmalloc-32
4 set_track() to save the alloc stack
2 Translate the object address to shadow address. shadow_addr = (addr >> 3) + KASAN_SHADOW_OFFSE.
3 Fill 2 bytes shadow memory with 0 and 1 byte with 4.Next byte is 0xFC(KASAN_KMALLOC_REDZONE).
32 bytes


kfree()
ARM64 MEMORY MAP
KASAN MAP
5 Read/write will cause a use-after-free bug.
object size(32 bytes)
shadow memory
1 free an object to slab cache(kmalloc-32).
kmalloc-32
4 set_track() to save the free stack
2 Translate the object address to shadow address. shadow_addr = (addr >> 3) + KASAN_SHADOW_OFFSE.
3 Fill the shadow memory with 0xFB(KASAN_KMALLOC_FREE) , where shadow memory size is 4 bytes(32B / 8).
32 bytes

KASAN

```
static noinline void __init kmalloc_oob_right(void)
{
 char *ptr;
 size_t size = 123;

 ptr = kmalloc(size, GFP_KERNEL);
 if (!ptr) {
  pr_err("Allocation failed\n");
  return;
 }

 ptr[size] = 'x';
 kfree(ptr);
}
```

```
==================================================================
BUG: KASAN: slab-out-of-bounds in kmalloc_oob_right+0x6c/0x8c
Write of size 1 at addr ffffffc0cb114d7b by task swapper/0/1

CPU: 4 PID: 1 Comm: swapper/0 Tainted: G S      W       4.9.82-perf+ #310
Hardware name: Qualcomm Technologies, Inc. SDM632 PMI632
Call trace:
[<ffffff90cf88d9f8>] dump_backtrace+0x0/0x320
[<ffffff90cf88dd2c>] show_stack+0x14/0x20
[<ffffff90cfdd1148>] dump_stack+0xa8/0xd0
......
[<ffffff90d0d6da70>] kernel_init+0x10/0x110
[<ffffff90cf8842a0>] ret_from_fork+0x10/0x30

Allocated by task 1:
 kasan_kmalloc+0xd8/0x188
 kmem_cache_alloc_trace+0x130/0x248
 ......
 kernel_init+0x10/0x110
 ret_from_fork+0x10/0x30

Freed by task 1:
 kasan_slab_free+0x88/0x178
 kfree+0x84/0x298
 kobject_uevent_env+0x144/0x620
 ......
 kernel_init+0x10/0x110
 ret_from_fork+0x10/0x30

The buggy address belongs to the object at ffffffc0cb114d00
 which belongs to the cache kmalloc-128 of size 128
The buggy address is located 123 bytes inside of
 128-byte region [ffffffc0cb114d00, ffffffc0cb114d80)
The buggy address belongs to the page:
page:ffffffbf032c4500 count:1 mapcount:0 mapping: (null) index:0xffffffc0cb115200 compound_mapcount: 0
flags: 0x4080(slab|head)
page dumped because: kasan: bad access detected

Memory state around the buggy address:
 ffffffc0cb114c00: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffffffc0cb114c80: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
>ffffffc0cb114d00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
                                                             ^
 ffffffc0cb114d80: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
 ffffffc0cb114e00: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
==================================================================
```

Write of size 1 at p[123], which result in out-of-bounds.

CONFIDENTIAL AND PROPRIETARY

Kmemleak

- Kmemleak tracks objects allocated via kmalloc/vmalloc/memblock.
- Kmemleak can not tracks the page allocations such as alloc_pages/_get_free_pages/ dma_alloc_coherent.
- Ioremap mappings are not tracked.
- Scanning the memory could take a long time (minutes)
  - Cannot lock the system during scanning
  - Memory allocation/freeing can still happen during scanning
- Kmemleak uses RCU list traversal to avoid locking

CONFIDENTIAL AND PROPRIETARY

Kmemleak

```c
void kmemleak_memalloc(void)
{
 char *pmem;

 pmem = kmalloc(300, GFP_KERNEL);
 if (!pmem)
 {
  printk("[Kmemleak]: kmalloc fail!\n");
 }
 else
 {
  printk("[Kmemleak]: kmalloc return %p \n", pmem);
 }
}

int __init kmemleak_test_init(void)
{

 printk("[Kmemleak]: kmemleak_test_init! \n");

 kmemleak_memalloc();

 return 0;
}

void __exit kmemleak_test_exit(void)
{
 printk("[Kmemleak]: kmemleak_test_exit now \n");
}
```

```
# echo clear > /sys/kernel/debug/kmemleak
... test the modules ...
# echo scan > /sys/kernel/debug/kmemleak
# cat /sys/kernel/debug/kmemleak
```

```
[ 1807.585257] [Kmemleak]: kmemleak_test_init!
[ 1807.585261] [Kmemleak]: kmalloc return ffff95c9bf81b600
```

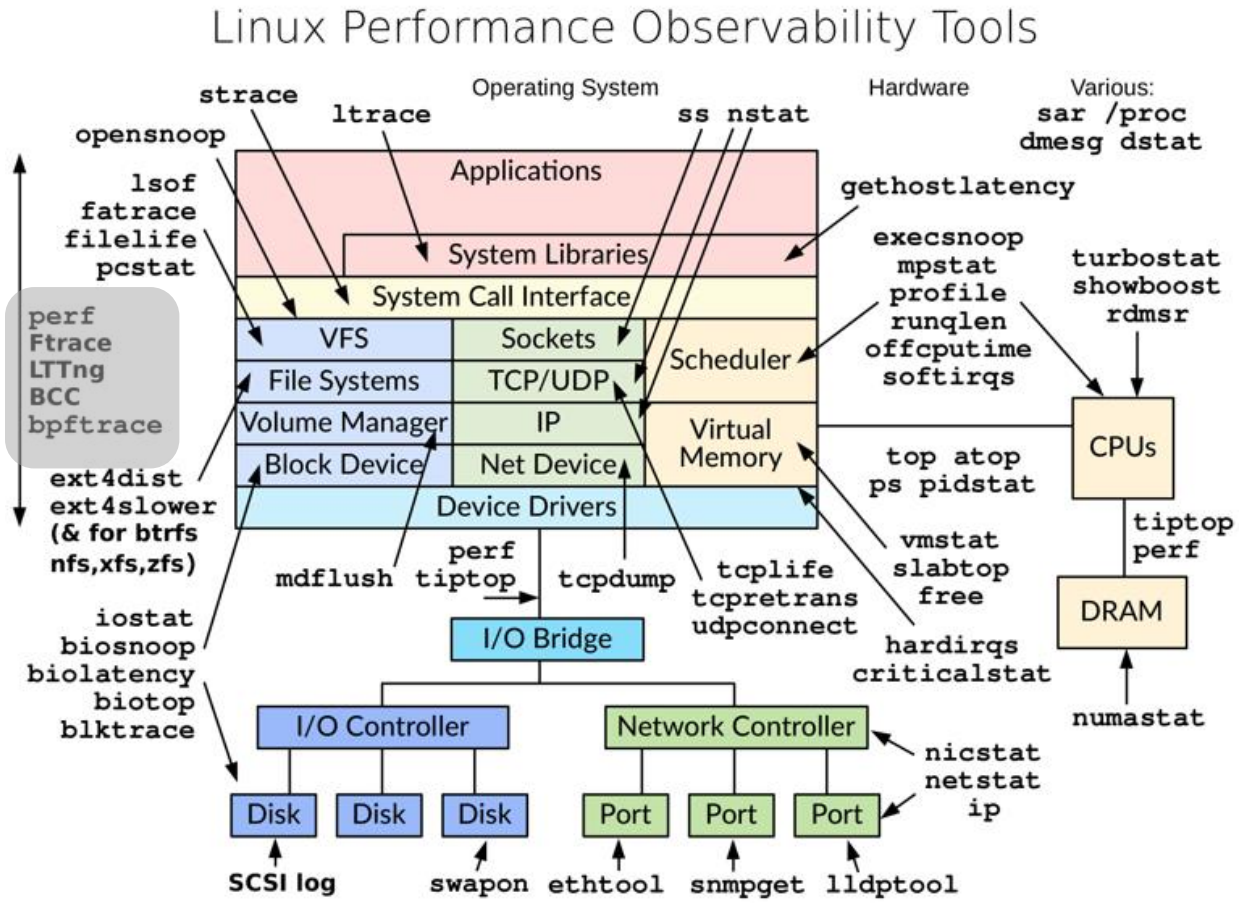
.....

```
unreferenced object 0xffff95c9bf81b600 (size 512):
  comm "insmod", pid 4882, jiffies 4295344192 (age 214.932s)
  hex dump (first 32 bytes):
    00 9e 81 bf c9 95 ff ff f0 29 67 10 00 ea ff ff  .........)g.....
    90 1b 67 10 00 ea ff ff 80 1c 67 10 00 ea ff ff  ..g.......g.....
  backtrace:
    [<ffffffffb378f6d5>] kmemleak_alloc+0x45/0xa0
    [<ffffffffb31a8363>] kmem_cache_alloc+0x103/0x1a0
    [<ffffffffc06ba015>] kmemleak_memalloc+0x15/0x39 [kmemleak_test]
    [<ffffffffc022d015>] 0xffffffffc022d015
    [<ffffffffb300218b>] do_one_initcall+0x4b/0x180
    [<ffffffffb3147155>] do_init_module+0x55/0x1dc
    [<ffffffffb30f19d1>] load_module+0x2291/0x2950
    [<ffffffffb30f22ef>] SYSC_finit_module+0xdf/0x110
    [<ffffffffb30f2339>] SyS_finit_module+0x9/0x10
    [<ffffffffb3799ab6>] entry_SYSCALL_64_fastpath+0x1e/0xa8
    [<ffffffffffffffff>] 0xffffffffffffffff
```

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

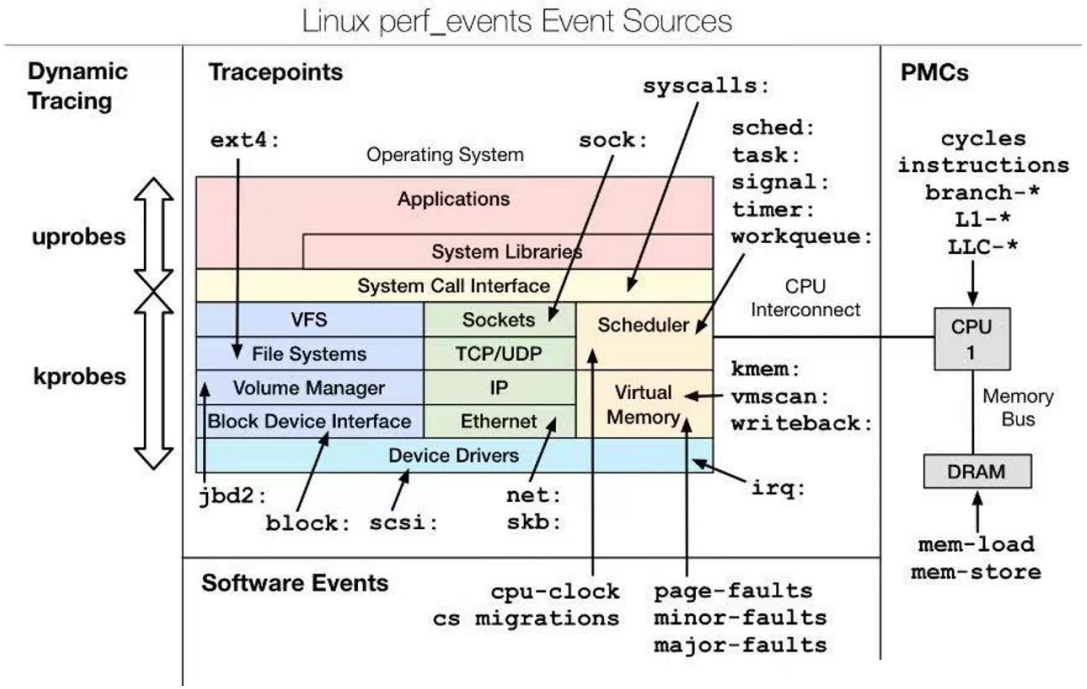
Performance Tools



CONFIDENTIAL AND PROPRIETARY

Perf

- perf_events
  - hardware events(PMU events)：cache-miss, ddr cycles
  - software events ： page-faults, cpu-migrations
  - tracepoint event



命令	作用
list	列出当前系统支持的所有性能事件。包括硬件性能事件、软件性能事件以及检查点。
test	perf对当前软硬件平台进行健全性测试，可用此工具测试当前的软硬件平台是否能支持perf的所有功能。
bench	perf中内置的benchmark，目前包括两套针对调度器和内存管理子系统的benchmark。
stat	执行某个命令，收集特定进程的性能概况，包括CPI、Cache丢失率等。
top	类似于linux的top命令，对系统性能进行实时分析。
probe	用于自定义动态检查点。
kmem	针对内核内存（slab）子系统进行追踪测量的工具
mem	内存存取情况
kvm	用来追踪测试运行在KVM虚拟机上的Guest OS。
lock	分析内核中的锁信息，包括锁的争用情况，等待延迟等。
sched	针对调度器子系统的分析工具。
trace	trace记录系统调用轨迹。
record	收集采样信息，并将其记录在数据文件中。随后可通过其它工具对数据文件进行分析。
report	读取perf record创建的数据文件，并给出热点分析结果。

Perf

- perf stat -I 1000 -a -e imx8_ddr0/cycles/,imx8_ddr0/write-cycles/,imx8_ddr0/read-cycles/

```
root@imx8mpevk:~# perf stat -I 1000 -a -e imx8_ddr0/cycles/,imx8_ddr0/write-cycles/,imx8_ddr0/read-cycles/
#           time             counts unit events
     1.000107250         20326876049      imx8_ddr0/cycles/
     1.000107250               21482      imx8_ddr0/write-cycles/
     1.000107250            93098992      imx8_ddr0/read-cycles/
     2.000740500          8918847862      imx8_ddr0/cycles/
     2.000740500                4268      imx8_ddr0/write-cycles/
     2.000740500            93092412      imx8_ddr0/read-cycles/
     3.001263000         12542319544      imx8_ddr0/cycles/
     3.001263000                1894      imx8_ddr0/write-cycles/
     3.001263000            93036097      imx8_ddr0/read-cycles/
     4.001758125          8918723649      imx8_ddr0/cycles/
     4.001758125                3797      imx8_ddr0/write-cycles/
     4.001758125            93077768      imx8_ddr0/read-cycles/
```

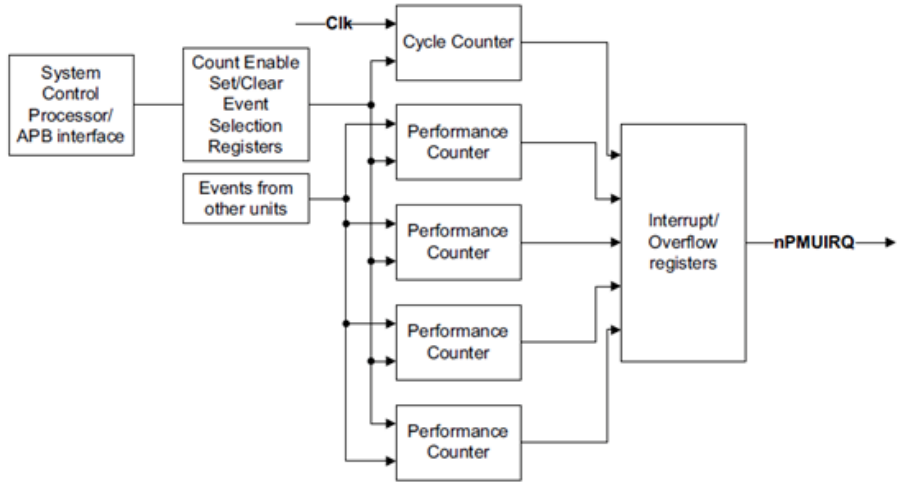
arch/arm64/kernel/perf_event.c

```
pmu {
        compatible = "arm,armv8-pmuv3";
        interrupts = <GIC_PPI 7
                (GIC_CPU_MASK_SIMPLE(6) | IRQ_TYPE_LEVEL_HIGH)>;
        interrupt-affinity = <&A53_0>, <&A53_1>, <&A53_2>, <&A53_3>;
};
```

drivers/perf/fsl_imx8_ddr_perf.c

```
ddr_pmu0: ddr-pmu@5c020000 {
                compatible = "fsl,imx8-ddr-pmu";
                reg = <0x5c020000 0x10000>;
                interrupts = <GIC_SPI 131 IRQ_TYPE_LEVEL_HIGH>;
};
```

Figure 11-1 shows the major blocks inside the PMU.

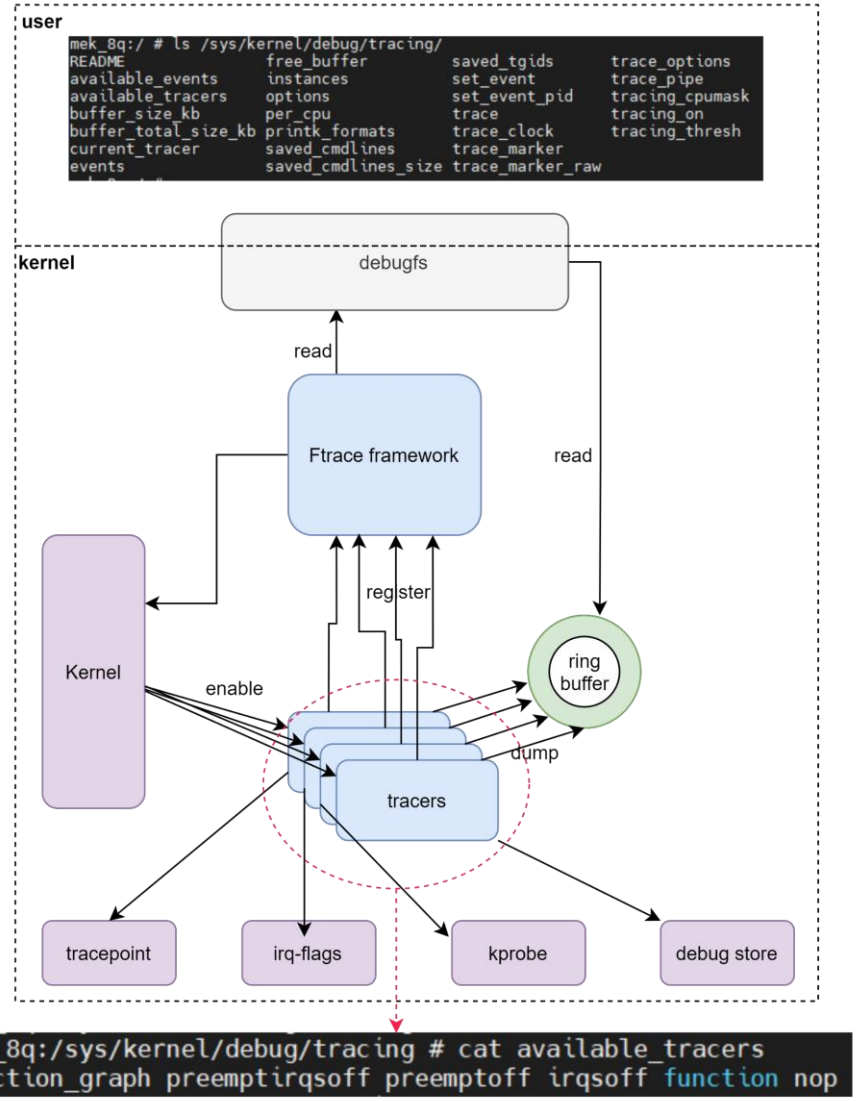


CONFIDENTIAL AND PROPRIETARY

Ftrace

Traces the internal operations of the kernel.

- Static **tracepoints** within the kernel (event tracing)
  - scheduling
  - interrupts
  - file systems
- Dynamic kernel function tracing
  - trace all functions within the kernel
  - call graphs
  - stack usage
- Latency tracers
  - how long interrupts are disabled (irqsoff)
  - how long preemption is disabled (preemptoff)
  - how long interrupts and/or preemption is disabled (preemptirqsoff)
  - how long it takes a process to run after it is woken (wakeup)



CONFIDENTIAL AND PROPRIETARY

Ftrace tracer

- function

```
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # echo function > current_tracer
mek_8q:/sys/kernel/debug/tracing # echo 1 > tracing_on
mek_8q:/sys/kernel/debug/tracing # run_test
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # cat trace | more
# tracer: function
#
#                              _-----> irqs-off
#                             / _----> need-resched
#                            | / _---> hardirq/softirq
#                            || / _--> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [003] d.h2  1470.353730: __rcu_read_unlock <-irq_find_mapping
          <idle>-0     [003] d.h2  1470.353732: generic_handle_irq <-__handle_domain_irq
          <idle>-0     [003] d.h2  1470.353733: handle_percpu_devid_irq <-generic_handle_irq
          <idle>-0     [003] d.h2  1470.353735: arch_timer_handler_phys <-handle_percpu_devid_irq
          <idle>-0     [003] d.h2  1470.353736: hrtimer_interrupt <-arch_timer_handler_phys
          <idle>-0     [003] d.h2  1470.353738: _raw_spin_lock <-hrtimer_interrupt
          <idle>-0     [003] d.h2  1470.353739: preempt_count_add <-_raw_spin_lo
```

- irqsoff

```
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # echo irqsoff > current_tracer
mek_8q:/sys/kernel/debug/tracing # echo 1 > tracing_on
mek_8q:/sys/kernel/debug/tracing # run_test
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # cat trace | more
# tracer: irqsoff
#
# irqsoff latency trace v1.1.5 on 4.14.98-dirty
# --------------------------------------------------------------------
# latency: 474 us, #183/183, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:6)
#    -----------------
#    | task: surfaceflinger-1935 (uid:1000 nice:-8 policy:1 rt_prio:1)
#    -----------------
#  => started at: el0_irq_naked
#  => ended at:   irq_exit
#
#
#                  _------> CPU#
#                 / _-----> irqs-off
#                | / _----> need-resched
#                || / _---> hardirq/softirq
#                ||| / _--> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
surfacef-1935    0d...    0us : el0_irq_naked
surfacef-1935    0d..1    2us : gic_handle_irq <-el0_irq_naked
surfacef-1935    0d..1    4us : __handle_domain_irq <-gic_handle_irq
surfacef-1935    0d..1    6us : irq_enter <-__handle_domain_irq
surfacef-1935    0d..1    7us : rcu_irq_enter <-irq_enter
surfacef-1935    0d..1    9us : preempt_count_add <-irq_enter
surfacef-1935    0d.h1   11us : irq_find_mapping <-__handle_domain_irq
```

- function_graph

```
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # echo function_graph > current_tracer
mek_8q:/sys/kernel/debug/tracing # echo 1 > tracing_on
mek_8q:/sys/kernel/debug/tracing # run_test
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # cat trace | more
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 1)   0.250 us    |                    preempt_count_add();
 1)   0.250 us    |                    preempt_count_sub();
 1)   5.125 us    |                  } /* kmem_cache_free */
 1) + 30.375 us   |                } /* put_cred_rcu */
 1)   0.125 us    |                rcu_cblist_dequeue();
 1)   0.250 us    |                rcu_segcblist_insert_done_cbs();
 1)   0.125 us    |                rcu_segcblist_insert_count();
 1)   0.125 us    |                rcu_segcblist_ready_cbs();
 1)   0.375 us    |                note_gp_changes();
 1)   0.250 us    |                cpu_needs_another_gp();
 1)   0.125 us    |                rcu_segcblist_ready_cbs();
```

- preemptoff

```
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # echo preemptoff > current_tracer
mek_8q:/sys/kernel/debug/tracing # echo 1 > tracing_on
mek_8q:/sys/kernel/debug/tracing # run_test
mek_8q:/sys/kernel/debug/tracing # echo 0 > tracing_on
mek_8q:/sys/kernel/debug/tracing # cat trace | more
# tracer: preemptoff
#
# preemptoff latency trace v1.1.5 on 4.14.98-dirty
# --------------------------------------------------------------------
# latency: 2948 us, #229/51551, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:6)
#    -----------------
#    | task: cat-3132 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#  => started at: unmap_page_range
#  => ended at:   unmap_page_range
#
#
#                  _------> CPU#
#                 / _-----> irqs-off
#                | / _----> need-resched
#                || / _---> hardirq/softirq
#                ||| / _--> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
    cat-3132    0...2 2586us : mark_page_accessed <-unmap_page_range
    cat-3132    0...2 2588us : page_remove_rmap <-unmap_page_range
    cat-3132    0...2 2589us : lock_page_memcg <-page_remove_rmap
    cat-3132    0...2 2591us : __rcu_read_lock <-lock_page_memcg
    cat-3132    0...2 2593us : PageHuge <-page_remove_rmap
    cat-3132    0...2 2594us : unlock_page_memcg <-page_remove_rmap
```


CONFIDENTIAL AND PROPRIETARY

Ftrace function tracer

$echo blk_update_request > /sys/kernel/debug/tracing/set_ftrace_filter
$echo function > /sys/kernel/debug/tracing/current_tracer

编译阶段

未开启 Ftrace

```
0000000000012ac <blk_update_request>:
    12ac:       d10183ff        sub     sp, sp, #0x60
    12b0:       a9017bfd        stp     x29, x30, [sp,#16]
    12b4:       910043fd        add     x29, sp, #0x10
    12b8:       a90253f3        stp     x19, x20, [sp,#32]
    12bc:       a9035bf5        stp     x21, x22, [sp,#48]
    12c0:       a90463f7        stp     x23, x24, [sp,#64]
    12c4:       f9002bf9        str     x25, [sp,#80]
    12c8:       aa0003f6        mov     x22, x0
    12cc:       53001c38        uxtb    w24, w1
    12d0:       2a0203f5        mov     w21, w2
    12d4:       2a1803e0        mov     w0, w24
    12d8:       94000000        bl      12c <blk_status_to_errno>
```

开启 Ftrace

```
0000000000003f10 <blk_update_request>:
    3f10:       d10183ff        sub     sp, sp, #0x60
    3f14:       a9017bfd        stp     x29, x30, [sp,#16]
    3f18:       910043fd        add     x29, sp, #0x10
    3f1c:       a90253f3        stp     x19, x20, [sp,#32]
    3f20:       a9035bf5        stp     x21, x22, [sp,#48]
    3f24:       a90463f7        stp     x23, x24, [sp,#64]
    3f28:       f9002bf9        str     x25, [sp,#80]
    3f2c:       aa0003f6        mov     x22, x0
    3f30:       53001c38        uxtb    w24, w1
    3f34:       2a0203f5        mov     w21, w2
    3f38:       aa1e03e0        mov     x0, x30
    3f3c:       94000000        bl      0 <_mcount>
```

链接阶段

```
ffff8000104e43c8 <blk_update_request>:
ffff8000104e43c8:       d10183ff        sub     sp, sp, #0x60
ffff8000104e43cc:       a9017bfd        stp     x29, x30, [sp,#16]
ffff8000104e43d0:       910043fd        add     x29, sp, #0x10
ffff8000104e43d4:       a90253f3        stp     x19, x20, [sp,#32]
ffff8000104e43d8:       a9035bf5        stp     x21, x22, [sp,#48]
ffff8000104e43dc:       a90463f7        stp     x23, x24, [sp,#64]
ffff8000104e43e0:       f9002bf9        str     x25, [sp,#80]
ffff8000104e43e4:       aa0003f6        mov     x22, x0
ffff8000104e43e8:       53001c38        uxtb    w24, w1
ffff8000104e43ec:       2a0203f5        mov     w21, w2
ffff8000104e43f0:       aa1e03e0        mov     x0, x30
ffff8000104e43f4:       97ed1fde        bl      ffff80001002c36c <_mcount>
ffff8000104e43f8:       2a1803e0        mov     w0, w24
ffff8000104e43fc:       97fff432        bl      ffff8000104e14c4 <blk_status_to_errno>
...
```

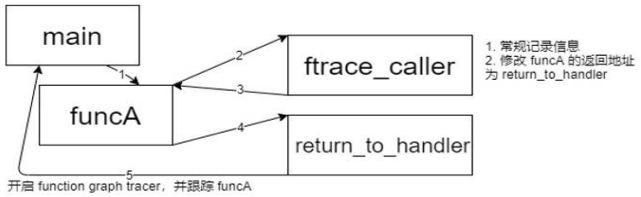
运行阶段

设定 trace 后

```
(gdb) x/20i blk_update_request
   0xffff8000104e43c8 <blk_update_request>:     sub     sp, sp, #0x60
   0xffff8000104e43cc <blk_update_request+4>:   stp     x29, x30, [sp,#16]
   0xffff8000104e43d0 <blk_update_request+8>:   add     x29, sp, #0x10
   0xffff8000104e43d4 <blk_update_request+12>:  stp     x19, x20, [sp,#32]
   0xffff8000104e43d8 <blk_update_request+16>:  stp     x21, x22, [sp,#48]
   0xffff8000104e43dc <blk_update_request+20>:  stp     x23, x24, [sp,#64]
   0xffff8000104e43e0 <blk_update_request+24>:  str     x25, [sp,#80]
   0xffff8000104e43e4 <blk_update_request+28>:  mov     x22, x0
   0xffff8000104e43e8 <blk_update_request+32>:  uxtb    w24, w1
   0xffff8000104e43ec <blk_update_request+36>:  mov     w21, w2
   0xffff8000104e43f0 <blk_update_request+40>:  mov     x0, x30
   0xffff8000104e43f4 <blk_update_request+44>:  bl      0xffff80001002c370 <ftrace_caller>
   0xffff8000104e43f8 <blk_update_request+48>:  mov     w0, w24
   0xffff8000104e43fc <blk_update_request+52>:  bl      0xffff8000104e14c4 <blk_status_to_errno>
```

ftrace_init 后

```
(gdb) x/20i blk_update_request
   0xffff8000104e43c8 <blk_update_request>:     sub     sp, sp, #0x60
   0xffff8000104e43cc <blk_update_request+4>:   stp     x29, x30, [sp,#16]
   0xffff8000104e43d0 <blk_update_request+8>:   add     x29, sp, #0x10
   0xffff8000104e43d4 <blk_update_request+12>:  stp     x19, x20, [sp,#32]
   0xffff8000104e43d8 <blk_update_request+16>:  stp     x21, x22, [sp,#48]
   0xffff8000104e43dc <blk_update_request+20>:  stp     x23, x24, [sp,#64]
   0xffff8000104e43e0 <blk_update_request+24>:  str     x25, [sp,#80]
   0xffff8000104e43e4 <blk_update_request+28>:  mov     x22, x0
   0xffff8000104e43e8 <blk_update_request+32>:  uxtb    w24, w1
   0xffff8000104e43ec <blk_update_request+36>:  mov     w21, w2
   0xffff8000104e43f0 <blk_update_request+40>:  mov     x0, x30
   0xffff8000104e43f4 <blk_update_request+44>:  nop
   0xffff8000104e43f8 <blk_update_request+48>:  mov     w0, w24
   0xffff8000104e43fc <blk_update_request+52>:  bl      0xffff8000104e14c4 <blk_status_to_errno>
```



流程说明：
1. main 调用 funcA

2. funcA 调用 ftrace_caller，ftrace_caller 记录信息，并将 funcA 的返回地址修改成 return_to_handler

3. ftrace_caller 返回，执行真正的 funcA 主体函数

4. funcA 主体执行完毕，执行返回指令，因为返回地址已经被修改，所以返回到 return_to_handler

5. return_to_handler 记录调试信息后返回，回复funcA的原始返回地址，并执行返回指令

CONFIDENTIAL AND PROPRIETARY

Ftrace kprobe

```
  p[:[GRP/]EVENT] [MOD:]SYM[+offs]|MEMADDR [FETCHARGS]  : Set a probe
  r[MAXACTIVE][:[GRP/]EVENT] [MOD:]SYM[+0] [FETCHARGS]  : Set a return probe
  -:[GRP/]EVENT                                         : Clear a probe

GRP            : Group name. If omitted, use "kprobes" for it.
EVENT          : Event name. If omitted, the event name is generated
                 based on SYM+offs or MEMADDR.
MOD            : Module name which has given SYM.
SYM[+offs]     : Symbol+offset where the probe is inserted.
MEMADDR        : Address where the probe is inserted.
MAXACTIVE      : Maximum number of instances of the specified function that
                 can be probed simultaneously, or 0 for the default value
                 as defined in Documentation/kprobes.txt section 1.3.1.

FETCHARGS      : Arguments. Each probe can have up to 128 args.
 %REG          : Fetch register REG
 @ADDR         : Fetch memory at ADDR (ADDR should be in kernel)
 @SYM[+|-offs] : Fetch memory at SYM +|- offs (SYM should be a data symbol)
 $stackN       : Fetch Nth entry of stack (N >= 0)
 $stack        : Fetch stack address.
 $retval       : Fetch return value.(*)
 $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. Currently, basic types
                 (u8/u16/u32/u64/s8/s16/s32/s64), hexadecimal types
                 (x8/x16/x32/x64), "string" and bitfield are supported.

 (*) only for return probe.
 (**) this is useful for fetching a field of data structures.
```

```
cd /sys/kernel/debug/tracing/
echo 'p:myprobe do_sys_open dfd=%ax filename=%dx flags=%cx mode=+4($stack)' > kprobe_events
echo 1 > tracing_on
echo 1 > events/kprobes/myprobe/enable
```

```
cat /sys/kernel/debug/tracing/events/kprobes/myprobe/format
name: myprobe
ID: 780
format:
        field:unsigned short common_type;       offset:0;       size:2; signed:0;
        field:unsigned char common_flags;       offset:2;       size:1; signed:0;
        field:unsigned char common_preempt_count;       offset:3; size:1;signed:0;
        field:int common_pid;   offset:4;       size:4; signed:1;

        field:unsigned long __probe_ip; offset:12;      size:4; signed:0;
        field:int __probe_nargs;        offset:16;      size:4; signed:1;
        field:unsigned long dfd;        offset:20;      size:4; signed:0;
        field:unsigned long filename;   offset:24;      size:4; signed:0;
        field:unsigned long flags;      offset:28;      size:4; signed:0;
        field:unsigned long mode;       offset:32;      size:4; signed:0;


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

```
cat /sys/kernel/debug/tracing/trace
# tracer: nop
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |<...>-1447  [001] 1038282.286875: myprobe: (do_sys_open+0x0/0xd6) dfd=3 filename=7f
           <...>-1447  [001] 1038282.286878: myretprobe: (sys_openat+0xc/0xe <- do_sys_open) $retval=fffffffffffffffe
           <...>-1447  [001] 1038282.286885: myprobe: (do_sys_open+0x0/0xd6) dfd=ffffff9c filename=40413c flags=8000 mode=1b6
           <...>-1447  [001] 1038282.286915: myretprobe: (sys_open+0x1b/0x1d <- do_sys_open) $retval=3
           <...>-1447  [001] 1038282.286969: myprobe: (do_sys_open+0x0/0xd6) dfd=ffffff9c filename=4041c6 flags=98800 mode=10
           <...>-1447  [001] 1038282.286976: myretprobe: (sys_open+0x1b/0x1d <- do_sys_open) $retval=3
```


CONFIDENTIAL AND PROPRIETARY

Ftrace front tool

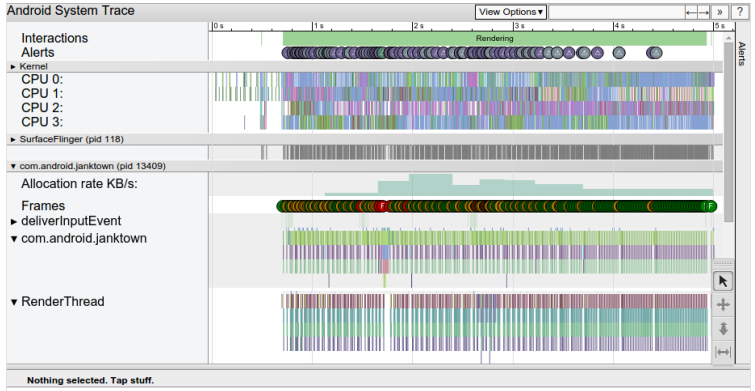
Binary tool to read Ftrace's buffers

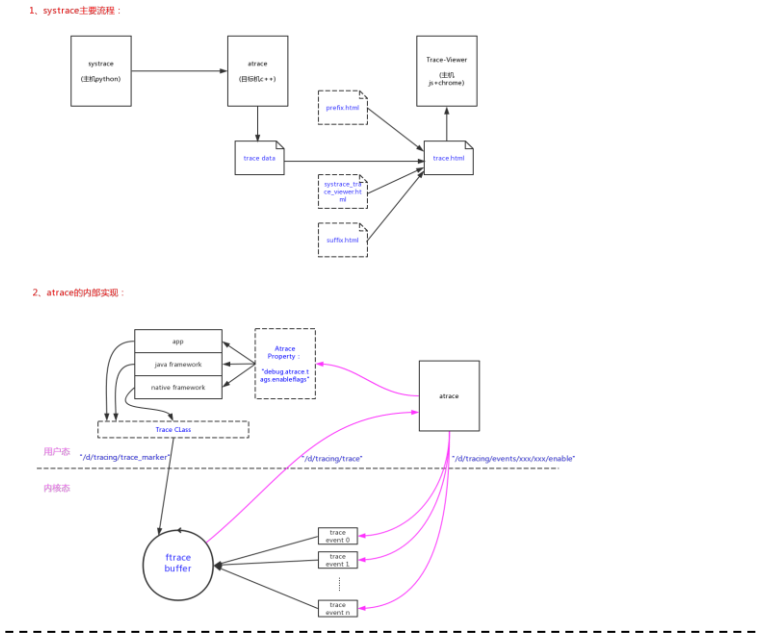
- trace-cmd

```
trace-cmd version 2.5.6

usage:
trace-cmd [COMMAND] ...

commands:
record - record a trace into a trace.dat file
start - start tracing without recording into a file
extract - extract a trace from the kernel
stop - stop the kernel from recording trace data
restart - restart the kernel trace data recording
show - show the contents of the kernel tracing buffer
reset - disable all kernel tracing and clear the trace buffers
report - read out the trace stored in a trace.dat file
stream - Start tracing and read the output directly
profile - Start profiling and read the output directly
hist - show a historgram of the trace.dat information
stat - show the status of the running tracing (ftrace) system
split - parse a trace.dat file into smaller file(s)
```

- kernelshark

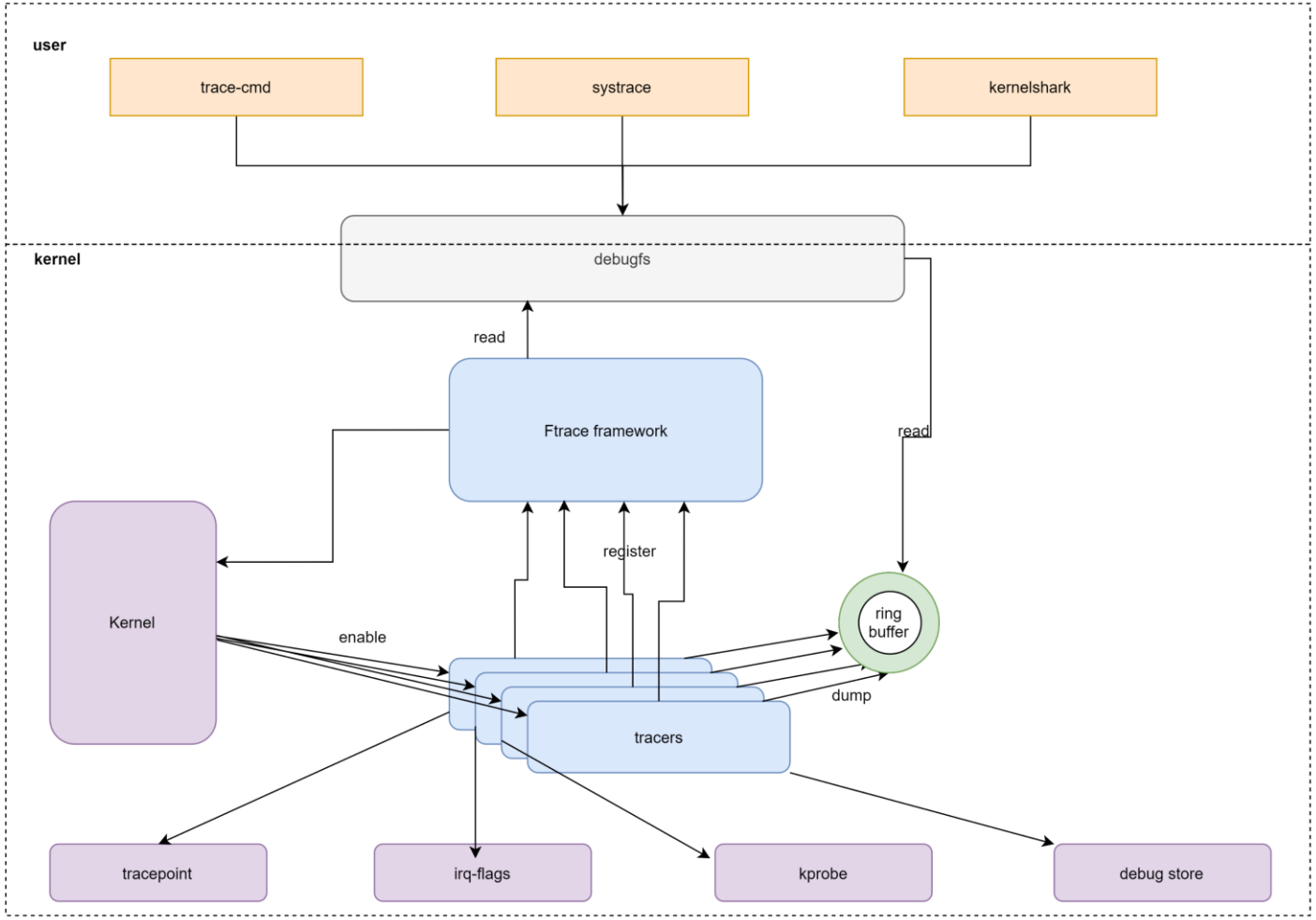
- systrace





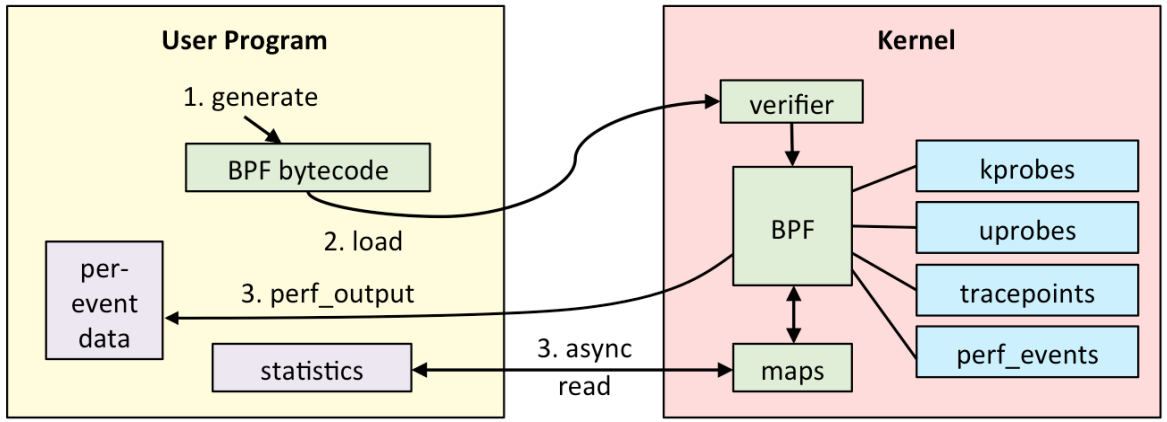
CONFIDENTIAL AND PROPRIETARY

Ftrace

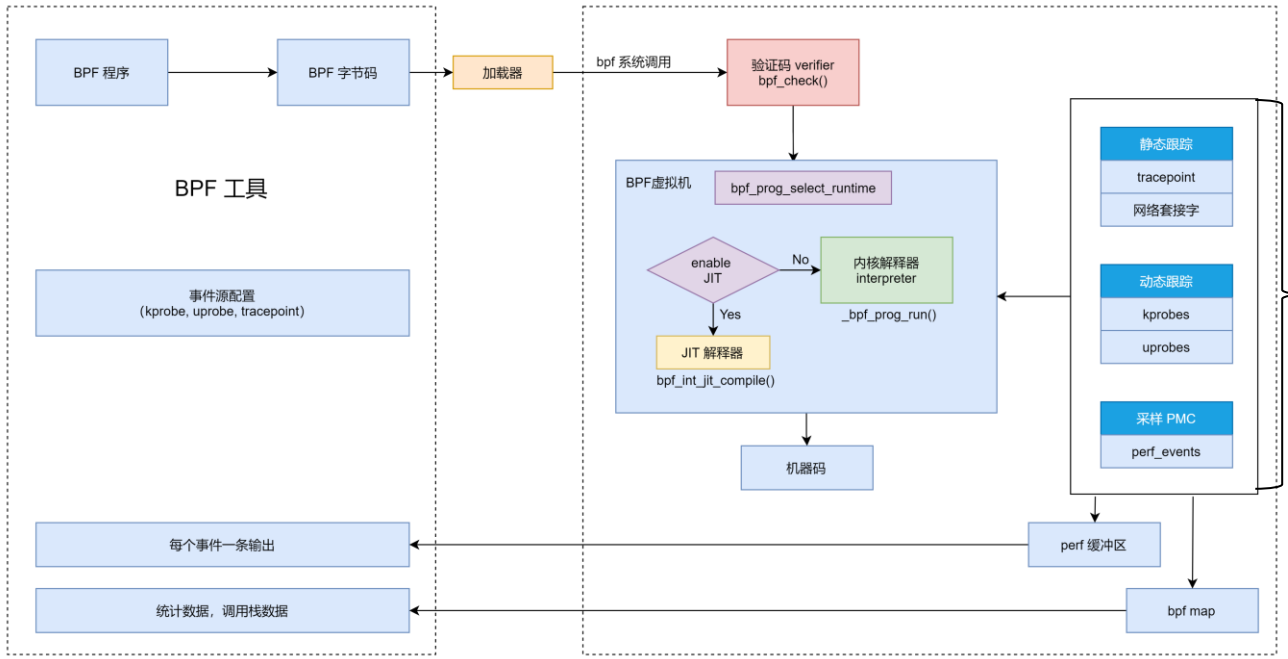


CONFIDENTIAL AND PROPRIETARY

eBPF



- Security
- Networking (XDP)
- Monitoring
- Tracing&Profiling



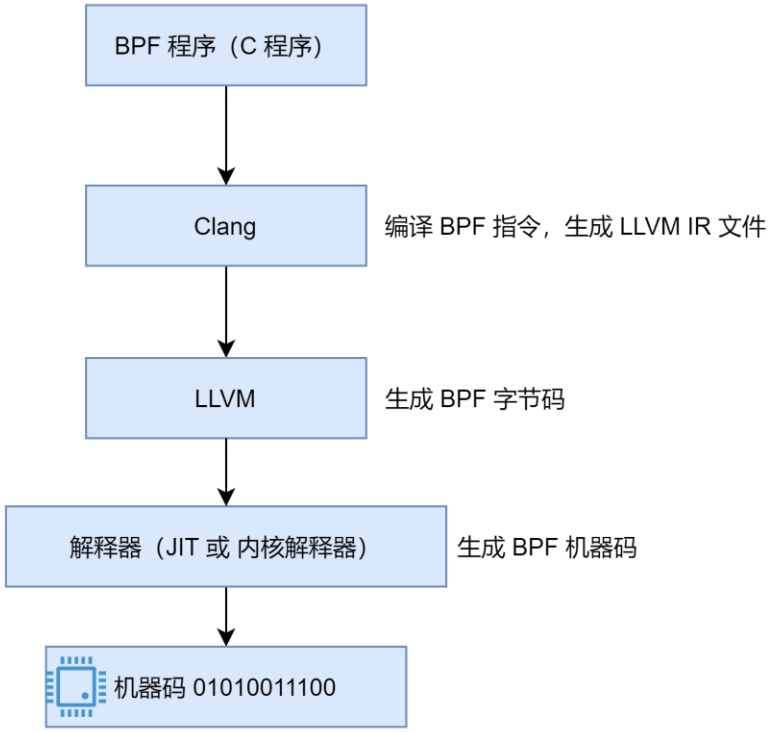
/include/uapi/linux/bpf.h

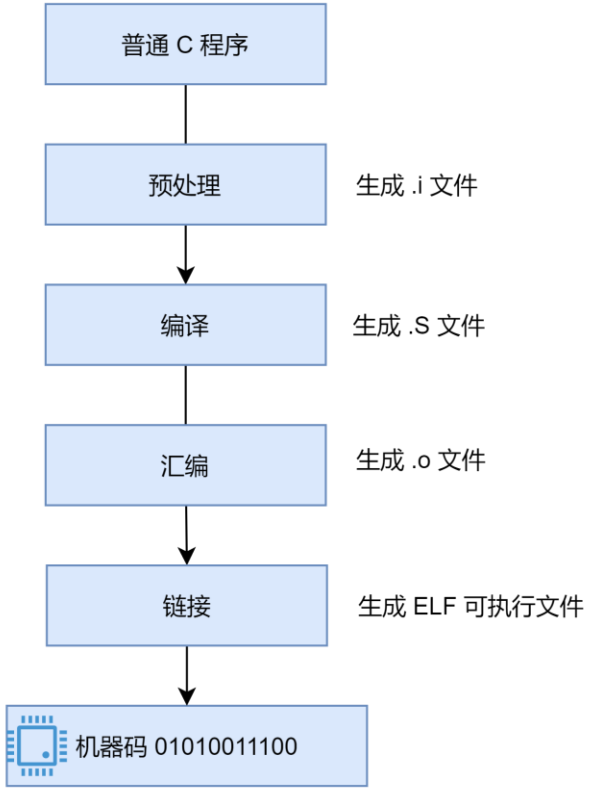
```
enum bpf_prog_type {
    BPF_PROG_TYPE_UNSPEC,
    BPF_PROG_TYPE_SOCKET_FILTER,
    BPF_PROG_TYPE_KPROBE,
    BPF_PROG_TYPE_SCHED_CLS,
    BPF_PROG_TYPE_SCHED_ACT,
    BPF_PROG_TYPE_TRACEPOINT,
    BPF_PROG_TYPE_XDP,
    BPF_PROG_TYPE_PERF_EVENT,
    BPF_PROG_TYPE_CGROUP_SKB,
    BPF_PROG_TYPE_CGROUP_SOCK,
    BPF_PROG_TYPE_LWT_IN,
    BPF_PROG_TYPE_LWT_OUT,
    BPF_PROG_TYPE_LWT_XMIT,
    BPF_PROG_TYPE_SOCK_OPS,
    BPF_PROG_TYPE_SK_SKB,
    BPF_PROG_TYPE_CGROUP_DEVICE,
};
```

BPF程序类型介绍

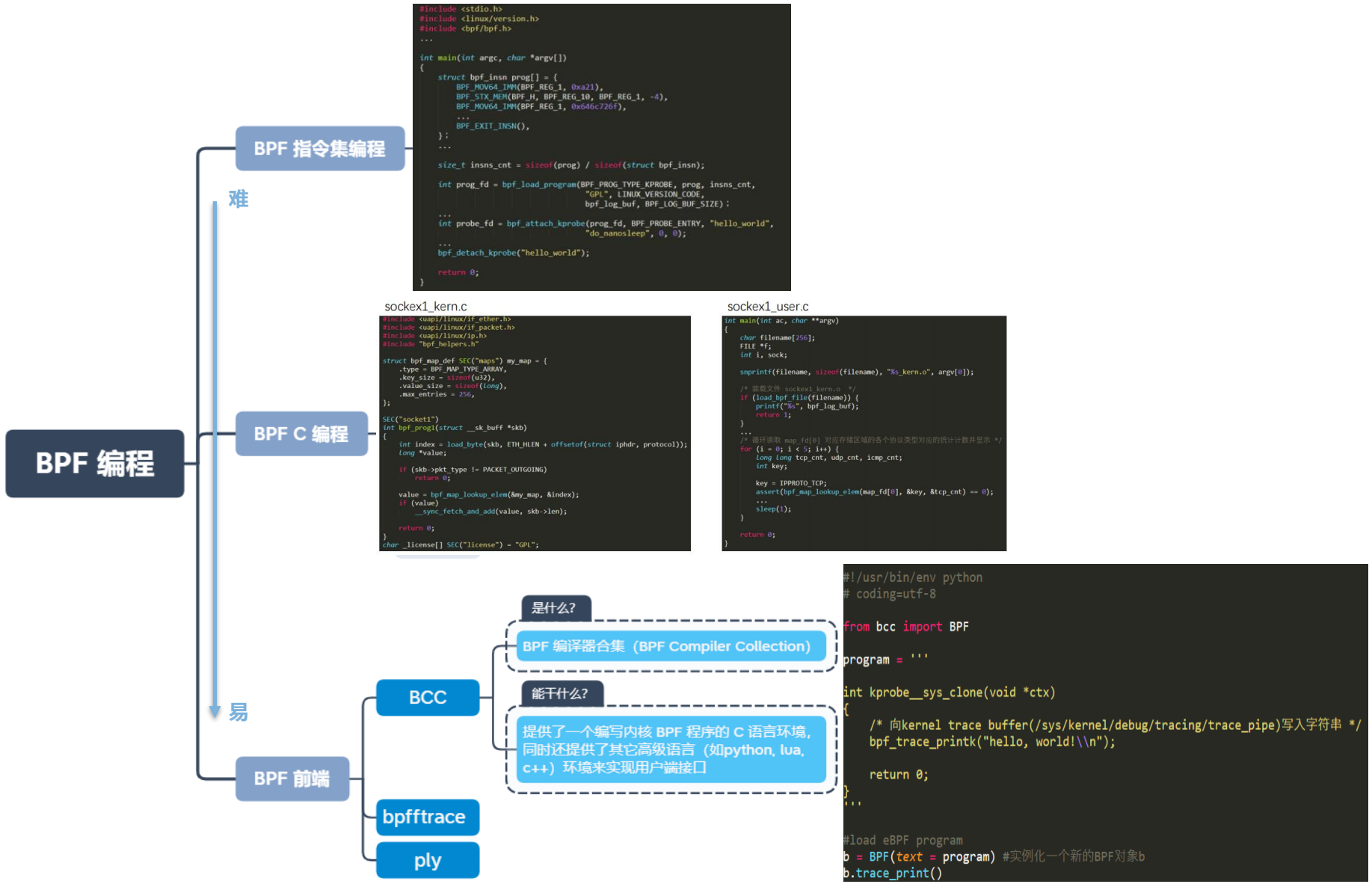
bpf_prog_type	描述
BPF_PROG_TYPE_KPROBE	用于内核动态插桩点kprobes
BPF_PROG_TYPE_TRACEPOINT	用于内核静态跟踪点
BPF_PROG_TYPE_PERF_EVENT	用于perf_events，包括PMC
…	…
BPF_PROG_TYPE_SOCKET_FILTER	用于挂载到网络套接字上(最早的BPF使用场景)
BPF_PROG_TYPE_SCHED_CLS	用于流量控制分类
BPF_PROG_TYPE_XDP	用于XDP(eXpress Data Path)程序
…	…

eBPF


BPF 程序（C 程序）
Clang
编译 BPF 指令，生成 LLVM IR 文件
LLVM
生成 BPF 字节码
解释器（JIT 或 内核解释器）
生成 BPF 机器码
机器码 01010011100


普通 C 程序
预处理
生成 .i 文件
编译
生成 .S 文件
汇编
生成 .o 文件
链接
生成 ELF 可执行文件
机器码 01010011100

eBPF tooling




```c
#include <stdio.h>
#include <linux/version.h>
#include <bpf/bpf.h>
...

int main(int argc, char *argv[])
{
    struct bpf_insn prog[] = {
        BPF_MOV64_IMM(BPF_REG_1, 0xa21),
        BPF_STX_MEM(BPF_H, BPF_REG_10, BPF_REG_1, -4),
        BPF_MOV64_IMM(BPF_REG_1, 0x646c726f),
        ...
        BPF_EXIT_INSN(),
    };
    ...

    size_t insns_cnt = sizeof(prog) / sizeof(struct bpf_insn);

    int prog_fd = bpf_load_program(BPF_PROG_TYPE_KPROBE, prog, insns_cnt,
                                   "GPL", LINUX_VERSION_CODE,
                                   bpf_log_buf, BPF_LOG_BUF_SIZE);
    ...
    int probe_fd = bpf_attach_kprobe(prog_fd, BPF_PROBE_ENTRY, "hello_world",
                                     "do_nanosleep", 0, 0);
    ...
    bpf_detach_kprobe("hello_world");

    return 0;
}
```

```c
#include <uapi/linux/if_ether.h>
#include <uapi/linux/if_packet.h>
#include <uapi/linux/ip.h>
#include "bpf_helpers.h"

struct bpf_map_def SEC("maps") my_map = {
    .type = BPF_MAP_TYPE_ARRAY,
    .key_size = sizeof(u32),
    .value_size = sizeof(long),
    .max_entries = 256,
};

SEC("socket1")
int bpf_prog1(struct __sk_buff *skb)
{
    int index = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
    long *value;

    if (skb->pkt_type != PACKET_OUTGOING)
        return 0;

    value = bpf_map_lookup_elem(&my_map, &index);
    if (value)
        __sync_fetch_and_add(value, skb->len);

    return 0;
}
char _license[] SEC("license") = "GPL";
```

```c
int main(int ac, char **argv)
{
    char filename[256];
    FILE *f;
    int i, sock;

    snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);

    /* 装载文件 sockex1_kern.o */
    if (load_bpf_file(filename)) {
        printf("%s", bpf_log_buf);
        return 1;
    }
    ...
    /* 循环读取 map_fd[0] 对应存储区域的各个协议类型对应的统计计数并显示 */
    for (i = 0; i < 5; i++) {
        long long tcp_cnt, udp_cnt, icmp_cnt;
        int key;

        key = IPPROTO_TCP;
        assert(bpf_map_lookup_elem(map_fd[0], &key, &tcp_cnt) == 0);
        ...
        sleep(1);
    }

    return 0;
}
```

```python
#!/usr/bin/env python
# coding=utf-8

from bcc import BPF

program = '''

int kprobe__sys_clone(void *ctx)
{
    /* 向kernel trace buffer(/sys/kernel/debug/tracing/trace_pipe)写入字符串 */
    bpf_trace_printk("hello, world!\\n");

    return 0;
}
'''

#load eBPF program
b = BPF(text = program) #实例化一个新的BPF对象b
b.trace_print()
```


CONFIDENTIAL AND PROPRIETARY

Agenda

- OS and System analysis
- Oops/Panic case
  - addr2line
  - objdump
  - gdb
- Pstore
- Kdump
- Memory debugging
  - SLAB
  - KASAN
  - Kmemleak
- Performance
  - Perf
  - Ftrace
  - eBPF/bcc
- Q&A

SECURE CONNECTIONS
FOR A SMARTER WORLD