

XNU 审计笔记

Brightiup

2018-03-28*

目录

1 系统调用	1
1.1 系统调用种类以及入口	1
2 文件系统	2
3 网络系统	2
4 内存管理	2
5 进程管理	2
6 Mach 相关	2
7 BSD 相关	2
8 IOKit	3
8.1 IOUserClient 对象生命周期	3
9 缓解策略	6

*最后更新日期: 2018 年 3 月 30 日

1 系统调用

1.1 系统调用种类以及入口

xnu 总共有四种系统调用，如表 1。

系统调用种类	函数入口
unix	hndl_unix_scall64
mach	hndl_mach_scall64
machdep	hndl_mdep_scall64
diagnostics	hndl_diag_scall64

表 1: xnu 系统调用种类

通过 `syscall` 指令可以直接进入这几个函数。`syscall` 指令的处理入口为 `hndl_syscall`，其实现如下：

```

1 // xnu/osfmk/x86_64/idt64.s
2 /*
3  * 64bit Tasks
4  * System call entries via syscall only:
5  *
6  * r15  x86_saved_state64_t
7  * rsp  kernel stack
8  *
9  * both rsp and r15 are 16-byte aligned
10 * interrupts disabled
11 * direction flag cleared
12 */
13
14 Entry(hndl_syscall)
15     TIME_TRAP_UENTRY
16
17     movq    %gs:CPU_ACTIVE_THREAD,%rcx /* get current thread */
18     movl    $-1, TH_IOTIER_OVERRIDE(%rcx) /* Reset IO tier override to -1 before handling
19     syscall */
20     movq    TH_TASK(%rcx),%rbx /* point to current task */
21
22     /* Check for active vtimers in the current task */
23     TASK_VTIMER_CHECK(%rbx,%rcx)
24
25     /*
26     * We can be here either for a mach, unix machdep or diag syscall,
27     * as indicated by the syscall class:
28     */
29     movl    R64_RAX(%r15), %eax /* syscall number/class */
30     movl    %eax, %edx
31     andl    $(SYSCALL_CLASS_MASK), %edx /* syscall class */
32     cmpl    $(SYSCALL_CLASS_MACH<<SYSCALL_CLASS_SHIFT), %edx
33     je      EXT(hndl_mach_scall64)
34     cmpl    $(SYSCALL_CLASS_UNIX<<SYSCALL_CLASS_SHIFT), %edx
35     je      EXT(hndl_unix_scall64)
36     cmpl    $(SYSCALL_CLASS_MDEP<<SYSCALL_CLASS_SHIFT), %edx
37     je      EXT(hndl_mdep_scall64)
38     cmpl    $(SYSCALL_CLASS_DIAG<<SYSCALL_CLASS_SHIFT), %edx
39     je      EXT(hndl_diag_scall64)

```

```

39
40  /* Syscall class unknown */
41  sti
42  CCALL3(i386_exception, $(EXC_SYSCALL), %rax, $1)
43  /* no return */

```

Listing 1: 系统调用入口

其中几种系统调用的分类序号为:

```

1  // xnu/osfmk/mach/i386/syscall_sw.h
2  #define SYSCALL_CLASS_NONE 0 /* Invalid */
3  #define SYSCALL_CLASS_MACH 1 /* Mach */
4  #define SYSCALL_CLASS_UNIX 2 /* Unix/BSD */
5  #define SYSCALL_CLASS_MDEP 3 /* Machine-dependent */
6  #define SYSCALL_CLASS_DIAG 4 /* Diagnostics */
7  #define SYSCALL_CLASS_IPC 5 /* Mach IPC */

```

Listing 2: 系统调用分类序号

系统调用号为 $SYSCALL_CLASS \ll 24 \mid dispatch_num$ 的组合,例如 $SYSCALL_CLASS$ 为 3, $dispatch_num$ 为 0 则将进入 `hv_task_trap` 并最终进入 `AppleHV.kext` 对应的处理函数,示例如下:

```

1  //
2  void trap_by_syscall_ins() {
3      asm ("mov $0x03000000, %rax; mov $0x04, %rdi; syscall")
4  }

```

Listing 3: 使用 syscall 指令

关于 GCC 和 clang 内嵌汇编参考:

- <http://cs.du.edu/~mitchell/bootcamp/x86-64.pdf>
- <https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html>

2 文件系统

3 网络系统

4 内存管理

5 进程管理

6 Mach 相关

7 BSD 相关

8 IOKit

8.1 IOUserClient 对象生命周期

IOUserClient 对象是用户态程序与驱动交互最主要的对象之一, 用户态通过 `IOServiceOpen` 函数可以打开一个驱动, 返回给用户态的为 `io_connect_t` 类型, 对应内核里类型为 **IKOT_IOKIT_CONNECT** 的 port 对象, 因而存储在该 port 里的真正对象为成员 `kdata.ip_kobject` 所对应的 IOUserClient 对象。

```

1 // device_server.c
2 /* Routine io_service_open_extended */
3 void _Xio_service_open_extended(mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
4 {
5     typedef struct {
6         mach_msg_header_t Head;
7         /* start of the kernel processed data */
8         mach_msg_body_t msgh_body;
9         mach_msg_port_descriptor_t owningTask;
10        mach_msgool_descriptor_t properties;
11        /* end of the kernel processed data */
12        NDR_record_t NDR;
13        uint32_t connect_type;
14        NDR_record_t ndr;
15        mach_msg_type_number_t propertiesCnt;
16        mach_msg_trailer_t trailer;
17    } Request __attribute__((unused));
18
19    Request *InOP = (Request *) InHeadP;
20    Reply *OutP = (Reply *) OutHeadP;
21
22    kern_return_t RetCode;
23    io_object_t service;
24    task_t owningTask;
25    io_connect_t connection;
26    .....
27    service = iokit_lookup_object_port(InOP->Head.msgh_request_port);
28    owningTask = convert_port_to_task(InOP->owningTask.name);
29    RetCode = is_io_service_open_extended(service, owningTask, InOP->connect_type,
30                                         InOP->ndr, (io_buf_ptr_t) (InOP->properties.address),
31                                         InOP->properties.size, &OutP->result, &connection);
32    task_deallocate(owningTask);
33    iokit_remove_reference(service);
34    if (RetCode != KERN_SUCCESS) {
35        MIG_RETURN_ERROR(OutP, RetCode);
36    }
37    .....
38    OutP->connection.name = (mach_port_t) iokit_make_connect_port(connection); <--- (a)
39    OutP->NDR = NDR_record;
40    OutP->Head.msgh_bits |= MACH_MSGH_BITS_COMPLEX;
41    OutP->Head.msgh_size = (mach_msg_size_t) (sizeof(Reply));
42    OutP->msgh_body.msgh_descriptor_count = 1;
43 }

```

Listing 4: `_Xio_service_open_extended` 函数

在 (a) 处, `iokit_make_connect_port` 函数将创建一个 **IKOT_IOKIT_CONNECT** 类型的 mach port, 最后调用 `iokit_alloc_object_port` 函数真正创建并且将 IOUserClient 对象与 port 关联。

```

1 // iokit_rpc.c
2 ipc_port_t iokit_make_connect_port(io_object_t obj) {
3     ipc_port_t port;
4     ipc_port_t sendPort;
5
6     if (obj == NULL) return IP_NULL;
7
8     port = iokit_port_for_object(obj, IKOT_IOKIT_CONNECT);
9     if (port) {
10         sendPort = ipc_port_make_send(port);
11         iokit_release_port(port);
12     } else
13         sendPort = IP_NULL;
14
15     iokit_remove_reference(obj);
16     return (sendPort);
17 }

```

Listing 5: iokit_make_connect_port 函数

```

1 // iokit_rpc.c
2 ipc_port_t iokit_alloc_object_port(io_object_t obj, ipc_kobject_type_t type) {
3     ipc_port_t notify;
4     ipc_port_t port;
5     do {
6         /* Allocate port, keeping a reference for it. */
7         port = ipc_port_alloc_kernel();
8         if (port == IP_NULL) continue;
9         /* set kobject & type */
10        // iokit_add_reference( obj );
11        ipc_kobject_set(port, (ipc_kobject_t)obj, type);
12        /* Request no-senders notifications on the port. */
13        ip_lock(port);
14        notify = ipc_port_make_sonce_locked(port);
15        ipc_port_nsrequest(port, 1, notify, &notify);
16        /* port unlocked */
17        assert(notify == IP_NULL);
18        gIOKitPortCount++;
19    } while (FALSE);
20    return (port);
21 }

```

Listing 6: iokit_alloc_object_port 函数

在此后一直到内核里的 IOUserClient 对象完全销毁之前,每次通过用户态的 io_connect_t 作为参数的所有 MIG¹函数进入内核后都会调用 iokit_lookup_connect_port 进行检查类型检查,并增加 retainCount。

```

1 // iokit_rpc.c
2 io_object_t iokit_lookup_connect_port(ipc_port_t port) {
3     io_object_t obj;
4     if (!IP_VALID(port)) return (NULL);
5     iokit_lock_port(port);
6     if (ip_active(port) && (ip_kotype(port) == IKOT_IOKIT_CONNECT)) {
7         obj = (io_object_t)port->ip_kobject;
8         iokit_add_connect_reference(obj);
9     } else
10        obj = NULL;

```

¹<http://www.cs.cmu.edu/afs/cs/project/mach/public/doc/unpublished/mig.ps>

```
11     iokit_unlock_port(port);  
12     return (obj);  
13 }
```

Listing 7: iokit_lookup_connect_port 函数

9 缓解策略