# 快速链接:

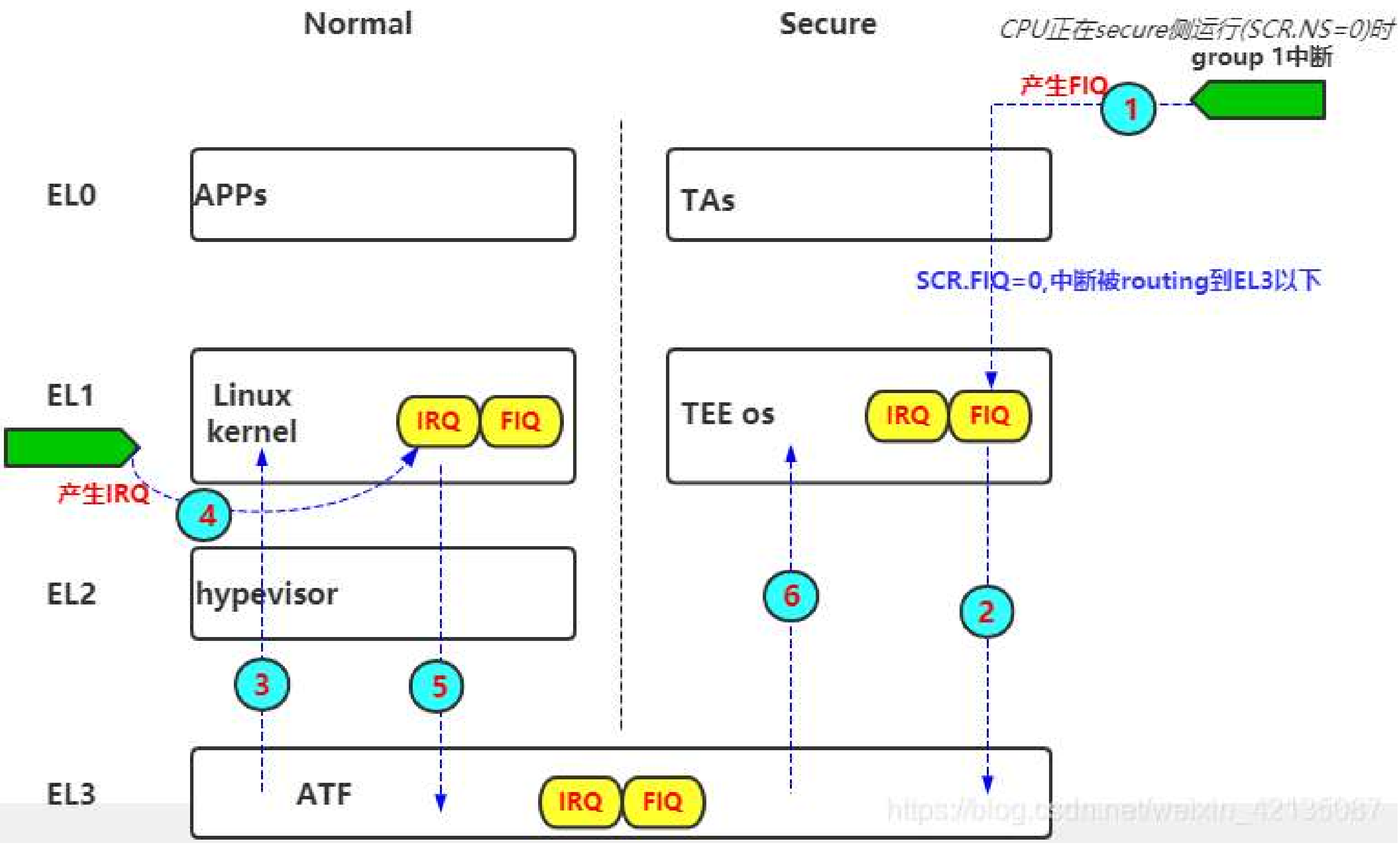
👉 👉 👉 ARMv8/ARMv9架构入门到精通-[目录] 👈 👈 👈

- 付费专栏-付费课程 **【购买须知】:**
- 联系方式-加入交流群 ----**联系方式-加入交流群**
- 个人博客笔记导读目录(全部)

---

> 环境:
> linux kernel 4.4, (SCR.IRQ=0、SCR.FIQ=1)
> optee 3.6 (SCR.IRQ=0、SCR.FIQ=0)
> ARMV8
> GICV3

当cpu处于secure侧时，来了一个非安全中断，根据SCR.NS=0/中断在non-secure group1组，cpu interface将会给cpu一个FIQ，(由于SCR.FIQ=0，FIQ将被routing到EL1),跳转至optee的fiq中断异常向量表,
再optee的fiq处理函数中，调用了smc跳转到ATF， ATF再切换至normal EL1(linux)， 此时SCR.NS的状态发生变化，根据SCR.NS=1/中断在non-secure group1组，cpu interface会再给cpu发送一个IRQ异常，
cpu跳转至linux的irq中断异常向量表，处理完毕后，再依次返回到ATF—返回到optee



我们从那代码中，依次拆解以上步骤：

**在cpu进入TEE之前，**CPU是通过optee_open_session()、optee_close_session()、optee_invoke_func()等函数进入TEE，而这些函数都是调用了optee_do_call_with_arg()，在该函数中再调用smc。

optee_do_call_with_arg()函数原型如下：（注意中文注释）

```
u32 optee_do_call_with_arg(struct tee_context *ctx, phys_addr_t parg)
{
    struct optee *optee = tee_get_drvdata(ctx->teedev);
    struct optee_call_waiter w;
    struct optee_rpc_param param = { };
    struct optee_call_ctx call_ctx = { };
    u32 ret;

    param.a0 = OPTEE_SMC_CALL_WITH_ARG;
    reg_pair_from_64(&param.a1, &param.a2, parg);
    /* Initialize waiter */
    optee_cq_wait_init(&optee->call_queue, &w);
    while (true) {
        struct arm_smccc_res res;

        // 注意，这里调用smc，-->ATF-->TEE
        optee->invoke_fn(param.a0, param.a1, param.a2, param.a3,
                 param.a4, param.a5, param.a6, param.a7,
                 &res);
        // 从TEE回来之后，执行这里(无论是TEE正常返回，还是RPC返回，还是中断切过来的)
        // 如果是REE中断导致TEE切过来的，那么这里已经触发irq了，带irq执行完毕后，程序继续
        // 注：REE中断导致的cpu从TEE切过来，也属于RPC返回

        if (res.a0 == OPTEE_SMC_RETURN_ETHREAD_LIMIT) {
            /*
             * Out of threads in secure world, wait for a thread
             * become available.
             */
            optee_cq_wait_for_completion(&optee->call_queue, &w);
        } else if (OPTEE_SMC_RETURN_IS_RPC(res.a0)) {
            // 如果是RPC、中断返回走这里，  然后看下optee_handle_rpc()函数原型
            param.a0 = res.a0;
            param.a1 = res.a1;
            param.a2 = res.a2;
            param.a3 = res.a3;
            optee_handle_rpc(ctx, &param, &call_ctx);
            // 如果是中断切过来的，optee_handle_rpc()函数相当于啥都没干，程序继续执行上面
            // 会调用optee->invoke_fn，cpu又切回了TEE
        } else {
            // 如果是正常返回，走这里，退出optee_do_call_with_arg()函数

```

```
            ret = res.a0;
            break;
        }
    }

    optee_rpc_finalize_call(&call_ctx);
    /*
     * We're done with our thread in secure world, if there's any
     * thread waiters wake up one.
     */
    optee_cq_wait_final(&optee->call_queue, &w);

    return ret;
}
```

**然后我们再看步骤1和步骤2：** 在optee中产生FIQ，跳转到FIQ中断向量表，然后调用smc切换到ATF

```
LOCAL_FUNC elx_irq , :
#if defined(CFG_ARM_GICV3)
    native_intr_handler irq
#else
    foreign_intr_handler    irq
#endif
END_FUNC elx_irq

LOCAL_FUNC elx_fiq , :
#if defined(CFG_ARM_GICV3)
    foreign_intr_handler    fiq  // 在optee运行时，来了REE中断，触发FIQ，程序会调用到这!
#else
    native_intr_handler fiq
#endif
END_FUNC elx_fiq
```

然后看下foreign_intr_handler 的具体实现：其实就是将当前进程的一些寄存器和栈寄存器保存，恢复中断模式的寄存器和tmp_stack栈，然后调用smc切换到ATF，其中smdid = TEESMC_OPTEED_RETURN_CALL_DONE

```
/* The handler of foreign interrupt. */
.macro foreign_intr_handler mode:req
    /*
     * Update core local flags
     */
    ldr w1, [sp, #THREAD_CORE_LOCAL_FLAGS]
    lsl w1, w1, #THREAD_CLF_SAVED_SHIFT
    orr w1, w1, #THREAD_CLF_TMP
    .ifc    \mode\(),fiq
    orr w1, w1, #THREAD_CLF_FIQ
    .else

```

```
    orr w1, w1, #THREAD_CLF_IRQ
    .endif
    str w1, [sp, #THREAD_CORE_LOCAL_FLAGS]

    /* get pointer to current thread context in x0 */
    get_thread_ctx sp, 0, 1, 2
    /* Keep original SP_EL0 */
    mrs x2, sp_el0

    /* Store original sp_el0 */
    str x2, [x0, #THREAD_CTX_REGS_SP]
    /* store x4..x30 */
    store_xregs x0, THREAD_CTX_REGS_X4, 4, 30
    /* Load original x0..x3 into x10..x13 */
    load_xregs sp, THREAD_CORE_LOCAL_X0, 10, 13
    /* Save original x0..x3 */
    store_xregs x0, THREAD_CTX_REGS_X0, 10, 13

    /* load tmp_stack_va_end */
    ldr x1, [sp, #THREAD_CORE_LOCAL_TMP_STACK_VA_END]
    /* Switch to SP_EL0 */
    msr spsel, #0
    mov sp, x1

    /*
     * Mark current thread as suspended
     */
    mov w0, #THREAD_FLAGS_EXIT_ON_FOREIGN_INTR
    mrs x1, spsr_el1
    mrs x2, elr_el1
    bl  thread_state_suspend
    mov w4, w0      /* Supply thread index */

    /* Update core local flags */
    /* Switch to SP_EL1 */
    msr spsel, #1
    ldr w0, [sp, #THREAD_CORE_LOCAL_FLAGS]
    lsr w0, w0, #THREAD_CLF_SAVED_SHIFT
    str w0, [sp, #THREAD_CORE_LOCAL_FLAGS]
    msr spsel, #0

    /*
     * Note that we're exiting with SP_EL0 selected since the entry
     * functions expects to have SP_EL0 selected with the tmp stack
     * set.
     */

    ldr w0, =TEESMC_OPTEED_RETURN_CALL_DONE
    ldr w1, =OPTEE_SMC_RETURN_RPC_FOREIGN_INTR
```

```
    mov w2, #0
    mov w3, #0
    /* w4 is already filled in above */
    smc #0
    b   .   /* SMC should not return */
.endm
```

**然后到了步骤3：**看ATF的opteed_smc_handler()函数，我们直接来看case TEESMC_OPTEED_RETURN_CALL_DONE处。

在optee时，触发了FIQ，foreign_intr_handler调用smc，进入ATF后，走这里，这里将恢复linux系统的寄存器，ELR_EL3填充linux侧的PC指针值，SMC_RET4后cpu将切回linux

```
    case TEESMC_OPTEED_RETURN_CALL_DONE:
        /*
         * This is the result from the secure client of an
         * earlier request. The results are in x0-x3. Copy it
         * into the non-secure context, save the secure state
         * and return to the non-secure state.
         */
        assert(handle == cm_get_context(SECURE));
        cm_el1_sysregs_context_save(SECURE);

        /* Get a reference to the non-secure context */
        ns_cpu_context = cm_get_context(NON_SECURE);
        assert(ns_cpu_context);

        /* Restore non-secure state */
        cm_el1_sysregs_context_restore(NON_SECURE);
        cm_set_next_eret_context(NON_SECURE);

        SMC_RET4(ns_cpu_context, x1, x2, x3, x4);
```

**接着又到了步骤4和步骤5：**该部分对应的代码就是本篇一开始贴出的optee_do_call_with_arg(), 程序回到次函数后，由于SCR.NS的状态发生了变化，cpu interface会再次给ARM Core发送一个IRQ，此时立即进入了linux kernel的IRQ中断向量表，待中断处理函数执行完毕后。PC再次指向此处，接着也就是下面这段逻辑了

```
else if (OPTEE_SMC_RETURN_IS_RPC(res.a0)) {
            // 如果是RPC、中断返回走这里， 然后看下optee_handle_rpc()函数原型
            param.a0 = res.a0;
            param.a1 = res.a1;
            param.a2 = res.a2;
            param.a3 = res.a3;
            optee_handle_rpc(ctx, &param, &call_ctx);
            // 如果是中断切过来的，optee_handle_rpc()函数相当于啥都没干，程序继续执行上面

```

```
            // 会调用optee->invoke_fn, cpu又切回了TEE
        }
```

在optee_handle_rpc()中的OPTEE_SMC_RPC_FUNC_FOREIGN_INTR业务逻辑中，其实啥逻辑都没干，直接返回. 子函数返回后，optee_do_call_with_arg()中的while循环继续执行，optee->invoke_fn()再次将CPU切到ATF。

```
void optee_handle_rpc(struct tee_context *ctx, struct optee_rpc_param *param,
              struct optee_call_ctx *call_ctx)
{
    struct tee_device *teedev = ctx->teedev;
    struct optee *optee = tee_get_drvdata(teedev);
    struct tee_shm *shm;
    phys_addr_t pa;

    switch (OPTEE_SMC_RETURN_GET_RPC_FUNC(param->a0)) {
    case OPTEE_SMC_RPC_FUNC_ALLOC:
        shm = tee_shm_alloc(ctx, param->a1, TEE_SHM_MAPPED);
        if (!IS_ERR(shm) && !tee_shm_get_pa(shm, 0, &pa)) {
            reg_pair_from_64(&param->a1, &param->a2, pa);
            reg_pair_from_64(&param->a4, &param->a5,
                     (unsigned long)shm);
        } else {
            param->a1 = 0;
            param->a2 = 0;
            param->a4 = 0;
            param->a5 = 0;
        }
        break;
    case OPTEE_SMC_RPC_FUNC_FREE:
        shm = reg_pair_to_ptr(param->a1, param->a2);
        tee_shm_free(shm);
        break;
    case OPTEE_SMC_RPC_FUNC_FOREIGN_INTR:  //---看下面的英文注释吧，如果是中断切过来的，
        /*
         * A foreign interrupt was raised while secure world was
         * executing, since they are handled in Linux a dummy RPC is
         * performed to let Linux take the interrupt through the normal
         * vector.
         */
        break;
    case OPTEE_SMC_RPC_FUNC_CMD:
        shm = reg_pair_to_ptr(param->a1, param->a2);
        handle_rpc_func_cmd(ctx, optee, shm, call_ctx);
        break;
    default:
        pr_warn("Unknown RPC func 0x%x\n",
            (u32)OPTEE_SMC_RETURN_GET_RPC_FUNC(param->a0));
```

```
            break;
        }

    param->a0 = OPTEE_SMC_CALL_RETURN_FROM_RPC;
}
```

**接下来步骤6：** 再次回到了ATF, 进入ATF的opteed_smc_handler()函数中，然后将optee_vectors->fast_smc_entry赋值给ELR_EL3，然后ERET退出ATF，跳转到optee中线程向量表的fast_smc_entry中

```
    if (is_caller_non_secure(flags)) {
        /*
         * This is a fresh request from the non-secure client.
         * The parameters are in x1 and x2. Figure out which
         * registers need to be preserved, save the non-secure
         * state and send the request to the secure payload.
         */
        assert(handle == cm_get_context(NON_SECURE));

        cm_el1_sysregs_context_save(NON_SECURE);

        /*
         * We are done stashing the non-secure context. Ask the
         * OPTEE to do the work now.
         */

        /*
         * Verify if there is a valid context to use, copy the
         * operation type and parameters to the secure context
         * and jump to the fast smc entry point in the secure
         * payload. Entry into S-EL1 will take place upon exit
         * from this function.
         */
        assert(&optee_ctx->cpu_ctx == cm_get_context(SECURE));

        /* Set appropriate entry for SMC.
         * We expect OPTEE to manage the PSTATE.I and PSTATE.F
         * flags as appropriate.
         */
        if (GET_SMC_TYPE(smc_fid) == SMC_TYPE_FAST) {
            cm_set_elr_el3(SECURE, (uint64_t)
                    &optee_vectors->fast_smc_entry);
        // linux处理完中断再回TEE时，走这里，将fast_smc_entry地址赋给了ELR_EL3， fast_sm
        // fast_smc_entry函数的地址，是optee开机初始化时，传过来的，然后ATF保存到全局变量
        }
```

**最后，**在optee的线程向量表的fast_smc_entry向量中，将恢复optee之前进程的寄存器和PC值，至此整个中断处理 流程完成。


Armv8/Armv9架构从入门到精通，Armv8/Armv9架构从入门到精通（一期），Armv8/Armv9架构从入门到精通（二期），Armv8/Armv9架构从入门到精通（三期），Arm一期、Arm二期、学习资料、免费、下载，全套资料，Secureboot从入门到精通，secureboot训练营，ATF架构从入门到精通、optee系统精讲、secureboot精讲，Trustzone/TEE/安全快速入门班，Trustzone/TEE/安全标准版，Trustzone/TEE/安全高配版。全套资料。周贺贺，baron，代码改变世界，coding_the_world，Arm精选，arm_2023，安全启动，加密启动

optee、ATF、TF-A、Trustzone、optee3.14、MMU、VMSA、cache、TLB、arm、armv8、armv9、TEE、安全、内存管理、页表，Non-cacheable,Cacheable, non-shareable,inner-shareable,outer-shareable, optee、ATF、TF-A、Trustzone、optee3.14、MMU、VMSA、cache、TLB、arm、armv8、armv9、TEE、安全、内存管理、页表…