linux中断

一、发生中断时的执行流程

> W(b)　vector_irq + stubs_offset -> vector_irq -> __irq_usr或者__irq_svc -> irq_handler -> gic_handle_irq -> handle_IRQ -> generic_handle_irq -> generic_handle_irq_desc -> desc->handle_irq

　　发生中断时，cpu进入irq异常模式，会跑到异常向量表运行。linux在之前已经初始化了异常向量表和异常处理代码，异常向量表在虚拟地址0xffff0000处。异常向量表和处理代码在arch/arm/kernel/entry_armv.S中定义。

```
...........................................................................
/* 异常处理函数 */
    .globl  __stubs_start
__stubs_start:  // 异常处理函数起始

    // irq、data abort、prefetch abort和udef异常处理函数都用vector_stub宏定义
    vector_stub irq, IRQ_MODE, 4

    .long   __irq_usr            @ 如果进入中断前是用户态则跳到__irq_usr
    .long   __irq_invalid        @  1
    .long   __irq_invalid        @  2
    .long   __irq_svc            @  如果进入中断前是内核态则跳到__irq_svc
    .long   __irq_invalid        @  4
    .long   __irq_invalid        @  5
    .long   __irq_invalid        @  6
    .long   __irq_invalid        @  7
    .long   __irq_invalid        @  8
    .long   __irq_invalid        @  9
    .long   __irq_invalid        @  a
    .long   __irq_invalid        @  b
    .long   __irq_invalid        @  c
    .long   __irq_invalid        @  d
    .long   __irq_invalid        @  e
    .long   __irq_invalid        @  f


    vector_stub dabt, ABT_MODE, 8

    .long   __dabt_usr       @  0  (USR_26 / USR_32)
    .long   __dabt_invalid       @  1  (FIQ_26 / FIQ_32)
    .long   __dabt_invalid       @  2  (IRQ_26 / IRQ_32)
    .long   __dabt_svc       @  3  (SVC_26 / SVC_32)
```

```
    .long   __dabt_invalid          @  4
...

// fiq、swi和address异常单独使用特定编写的处理函数
vector_fiq:
    subs    pc, lr, #4

vector_addrexcptn:
    b   vector_addrexcptn

    .align  5

.LCvswi:
    .word   vector_swi

    .globl  __stubs_end
__stubs_end:    /* 异常处理函数结束 */
.....................................................................................
/* 异常向量表 */
     .equ   stubs_offset, __vectors_start + 0x200 - __stubs_start
    .globl  __vectors_start
__vectors_start:    // 异常向量表起始
 ARM(   swi SYS_ERROR0  )
...
    W(b)    vector_und + stubs_offset
    W(ldr)  pc, .LCvswi + stubs_offset
    W(b)    vector_pabt + stubs_offset
    W(b)    vector_dabt + stubs_offset
    W(b)    vector_addrexcptn + stubs_offset
    W(b)    vector_irq + stubs_offset
    W(b)    vector_fiq + stubs_offset

    .globl  __vectors_end
__vectors_end:  // 异常向量表结束
...................................................................................
/* 除了fiq、swi和address异常之外的异常处理函数宏 */
vector_\name:
    .if \correction
    sub lr, lr, #\correction        //修正返回地址lr_<exception>
    .endif

    @
    @ Save r0, lr_<exception> (parent PC) and spsr_<exception>
    @ (parent CPSR)
    @
    /* 将r0，修正后的lr_<exception>，spsr_<exception>保存到sp_<exception>中，
     * spsr_<exception>是进入中断前的cpsr，sp_<exception>空间很小
     */
    stmia   sp, {r0, lr}        @ save r0, lr
    mrs lr, spsr            @ lr = spsr_<exception>
    str lr, [sp, #8]        @ save spsr

    @
```

```
    @ Prepare for SVC32 mode.  IRQs remain disabled.
    @
    /* 将cpsr_<exception>读到r0中，然后将模式域切换为SVC模式，
     * 再将r0写到spsr_<exception>，因此现在还是<exception>不是svc，
     * 同时未更改前的spsr_<exception>也保存着副本在sp_<exception>中
     */
    mrs r0, cpsr
    eor r0, r0, #(\mode ^ SVC_MODE | PSR_ISETSTATE)
    msr spsr_cxsf, r0

    @
    @ the branch table must immediately follow this code
    @
    /*
     * 保存sp_<exception>到r0，
     * 根据发生中断前的模式进入对应的子处理程序，如irq模式下的__irq_usr或者__irq_svc，
     * 同时将以修改的spsr_<exception>恢复到cpsr中，因此等于将中断前的cpsr的模式域切换为svc再恢复
     */
    and lr, lr, #0x0f   //获取修改前的spsr_<exception>即中断前的cpsr的模式
    mov r0, sp        //将sp_<exception>保存到r0，然后可以在子程序中使用
    ldr lr, [pc, lr, lsl #2]        //lr = pc + lr << 2
    movs    pc, lr      // 根据lr跳转到适合的子处理程序如irq模式下的__irq_usr或者__irq_svc，
                        //因为movs，所以同时cpsr = 修改过的spsr_<exception>
ENDPROC(vector_\name)

    .align  2
    @ handler addresses follow this label
1:
    .endm
.............................................................................
    .align  5
__irq_usr:
    usr_entry       // 保存usr中断现场
    kuser_cmpxchg_check
    irq_handler     //
    get_thread_info tsk     //
    mov why, #0             //why = 0
    b   ret_to_user_from_irq    //
...
ENDPROC(__irq_usr)
....................................................................................
    .align  5
__irq_svc:
    svc_entry       // 保存svc中断现场
    irq_handler     // 调用gic_handle_irq

#ifdef CONFIG_PREEMPT
    get_thread_info tsk     //
    ldr r8, [tsk, #TI_PREEMPT]      @ get preempt count
    ldr r0, [tsk, #TI_FLAGS]        @ get flags
    teq r8, #0              @ if preempt count != 0
    movne   r0, #0              @ force flags to 0
    tst r0, #_TIF_NEED_RESCHED
```

```
    blne    svc_preempt     //
#endif

...
    svc_exit r5             @ return from exception
...
ENDPROC(__irq_svc)
.................................................................................
    .macro  irq_handler
#ifdef CONFIG_MULTI_IRQ_HANDLER
    ldr r1, =handle_arch_irq    //对于socfpga会设置为gic_handle_irq
    mov r0, sp              //传给gic_handle_irq的参数r0 = sp_svc，即pt_regs
    adr lr, BSYM(9997f)
    ldr pc, [r1]            //调用gic_handle_irq
#else
    arch_irq_handler_default
#endif
9997:
    .endm
```

位于__vectors_start和__vectors_end之间的是真正的异常向量表，位于__stubs_start和__stubs_end之间的是处理代码。对于irq中断先进入W(b) vector_irq + stubs_offset然后进入vector_irq，根据进入irq前的模式进入__irq_usr或者__irq_svc，无论是__irq_usr还是__irq_svc都会进入irq_handler，然后调用handle_arch_irq即gic_handle_irq。其中__irq_usr和__irq_svc分别是保存从用户和内核态进入中断的保存中断现场。

```
    .macro  svc_entry, stack_hole=0
...
     /* sp_svc = sp_svc - (S_FRAME_SIZE  - 4), S_FRAME_SIZE是struct pt_regs的大小
      *
      */
     //sp_svc指向pt_regs中的ARM_r1
    sub sp, sp, #(S_FRAME_SIZE + \stack_hole - 4)
...
    stmia   sp, {r1 - r12}  //将r1到r12保存到sp_svc中，因为r1到r12在vector_\name中没有改变，因此
                          //这里的r1到r12是发生中断异常前的r1到r12
    ldmia   r0, {r3 - r5}   //r0保存着sp_<exception>，因此这里把vector_\name中保存到
sp_<exception>
                          //中的r0,lr_<exception>，spsr_<exception>恢复到r3,r4,r5
    /* S_SP是offsetof(struct pt_regs, ARM_sp)
    *  将pt_regs中ARM_sp处的地址给r7
    */
    add r7, sp, #S_SP - 4   @ here for interlock avoidance
    mov r6, #-1             @  ""  ""       ""        ""
    add r2, sp, #(S_FRAME_SIZE + \stack_hole - 4)   //r2保存原来未发生异常前的sp_svc
...
    // 将发生中断异常前的r0保存到sp_svc中的pt_regs中
    str r3, [sp, #-4]!      @ save the "real" r0 copied
                        @ from the exception stack
    mov r3, lr  //将lr_svc保存在r3中

    @
    @ We are now ready to fill in the remaining blanks on the stack:
```

```
    @
    @  r2 - sp_svc
    @  r3 - lr_svc
    @  r4 - lr_<exception>, already fixed up for correct return/restart
    @  r5 - spsr_<exception>
    @  r6 - orig_r0 (see pt_regs definition in ptrace.h)
    @
    stmia   r7, {r2 - r6}   //将最后几个保存到pt_regs中

...
    .endm
........................................
    .macro  usr_entry
...
    sub sp, sp, #S_FRAME_SIZE   //sp_svc指向pt_regs中的ARM_r0
    stmib   sp, {r1 - r12}      //将发生中断前的r1到r12保存到sp_svc的pt_regs中
    //将中断前的r0，pc（lr_<exception>）和cpsr(spsr_<exception>)恢复到r3,r4,r5
    ldmia   r0, {r3 - r5}
    //r0指向pt_regs中的ARM_pc
    add r0, sp, #S_PC       @ here for interlock avoidance
    mov r6, #-1         @  ""  ""      ""        ""
    // 将发生中断异常前的r0保存到pt_regs中
    str r3, [sp]        @ save the "real" r0 copied
                @ from the exception stack

    @
    @ We are now ready to fill in the remaining blanks on the stack:
    @
    @  r4 - lr_<exception>, already fixed up for correct return/restart
    @  r5 - spsr_<exception>
    @  r6 - orig_r0 (see pt_regs definition in ptrace.h)
    @
    @ Also, separately save sp_usr and lr_usr
    @
    stmia   r0, {r4 - r6}   //将这几个保存到pt_regs中，和svc不同的是没有保存sp_usr和lr_usr,放到下一步

    stmdb   r0, {sp, lr}^    //将sp_usr和lr_usr保存到pt_regs中，因为^，所以不是sp_svc和lr_svc
...
    .endm
```

看了保存中断现场，现在来看看恢复中断现场。

```
..................................................................
    /* 恢复svc中断现场 */
    .macro  svc_exit, rpsr
    msr spsr_cxsf, \rpsr    //将spsr_<exception>恢复，因为传参是r5
    ...
    //恢复所有的pt_regs，同时spsr_<exception>写到cpsr，如果发生中断前中断是打开，那么
    ldmia   sp, {r0 - pc}^          @ load r0 - pc, cpsr
    .endm
..................................................................
    /* 恢复usr中断现场 */
```

```
    ENTRY(ret_to_user_from_irq)
    ldr r1, [tsk, #TI_FLAGS]    //从svc栈的thread_info获取发生中断前的进程的task_struct中的flags
    tst r1, #_TIF_WORK_MASK     //获取flags中的work部分
    bne work_pending    //如果不为0说明有工作要做(有信号或者要被调度)
no_work_pending:
...
    //如果没有工作就正常恢复用户中断现场
    arch_ret_to_user r1, lr //什么都不做
    restore_user_regs fast = 0, offset = 0  //将保存的用户寄存器恢复
ENDPROC(ret_to_user_from_irq)



    /* 将保存的用户寄存器恢复 */
    .macro  restore_user_regs, fast = 0, offset = 0
    ldr r1, [sp, #\offset + S_PSR]  //获取发生异常前的cpsr到r1
    ldr lr, [sp, #\offset + S_PC]!  //获取异常后返回的地址到Lr_svc，同时更新sp_svc指向
    msr spsr_cxsf, r1           //将发生异常前的cpsr写到spsr_svc中
    ...
    .if \fast
    ldmdb   sp, {r1 - lr}^          @ get calling r1 - lr
    .else
    ldmdb   sp, {r0 - lr}^          //将sp_svc指向的pt_regs恢复到r0到r12以及sp_usr和lr_usr
    .endif
    mov r0, r0              @ ARMv5T and earlier require a nop
                        @ after ldm {}^
    add sp, sp, #S_FRAME_SIZE - S_PC    //将sp_svc指回没发生异常前的地方
    movs    pc, lr              //返回发生异常前的地址，同时恢复异常前的cpsr
    .endm
..................................................................
/* 处理pending事物,调度或者处理信号 */
work_pending:
    mov r0, sp              @ 'regs'
    mov r2, why             @ 'syscall'
    bl  do_work_pending     //调度或者处理信号（没打开中断，为什么能调度？因为中断已经处理完）
    cmp r0, #0  //如果do_work_pending返回值为0则跳到no_work_pending，否则要restar还是strace什么的
    beq no_work_pending
    movlt   scno, #(__NR_restart_syscall - __NR_SYSCALL_BASE)
    ldmia   sp, {r0 - r6}           @ have to reload r0 - r6
    b   local_restart           @ ... and off we go
..................................................................
/* 调度或处理信号 */
asmlinkage int do_work_pending(struct pt_regs *regs, unsigned int thread_flags, int
syscall)
{
    do {
        if (likely(thread_flags & _TIF_NEED_RESCHED)) { //调度
            schedule();
        } else {        //处理信号
            if (unlikely(!user_mode(regs)))
                return 0;
            local_irq_enable();
            if (thread_flags & _TIF_SIGPENDING) {
                int restart = do_signal(regs, syscall);
```

```
                if (unlikely(restart)) {
                    /*
                     * Restart without handlers.
                     * Deal with it without leaving
                     * the kernel space.
                     */
                    return restart;
                }
                syscall = 0;
            } else {
                clear_thread_flag(TIF_NOTIFY_RESUME);
                tracehook_notify_resume(regs);
            }
        }
        local_irq_disable();
        thread_flags = current_thread_info()->flags;
    } while (thread_flags & _TIF_WORK_MASK);
    return 0;
}
```

```
asmlinkage void __exception_irq_entry gic_handle_irq(struct pt_regs *regs)
{
    u32 irqstat, irqnr;
    struct gic_chip_data *gic = &gic_data[0];
    void __iomem *cpu_base = gic_data_cpu_base(gic);

    do {
        irqstat = readl_relaxed(cpu_base + GIC_CPU_INTACK);
        irqnr = irqstat & ~0x1c00;  /* 读出硬件中断号 */
/*
***************************************************************************
*                                    Embest Tech co., ltd
*                                    www.embest-tech.com
***************************************************************************
*
*chage the IPI interrupt handler, cause the another cpu not secheule the linux thread any
more.
*/
        if (likely(irqnr < 1021)) {
            if(irqnr<16)
            {
                /* 如果是sgi中断则写EOI */
                writel_relaxed(irqstat, cpu_base + GIC_CPU_EOI);
            }
            irqnr = irq_find_mapping(gic->domain, irqnr);   /* 硬件中断号转化为虚拟中断号 */
            handle_IRQ(irqnr, regs);    /* 进入irqnr的通用层处理 */
            continue;
        }
```

```
        break;
    } while (1);
}
..........................................................................
void handle_IRQ(unsigned int irq, struct pt_regs *regs)
{
    struct pt_regs *old_regs = set_irq_regs(regs);

    irq_enter();    /* 增加preempt count HARDOFF */

    if (unlikely(irq >= nr_irqs)) {
        if (printk_ratelimit())
            printk(KERN_WARNING "Bad IRQ%u\n", irq);
        ack_bad_irq(irq);
    } else {
        generic_handle_irq(irq);    /* 进入通用层 */
    }

    irq_exit();     /* 减少preempt count HARDOFF，同时检查是否可以和需要执行软中断 */
    set_irq_regs(old_regs);
}
...................................................................................
int generic_handle_irq(unsigned int irq)
{
    struct irq_desc *desc = irq_to_desc(irq); //取出irq对应的irq_desc

    if (!desc)
        return -EINVAL;
    generic_handle_irq_desc(irq, desc);
    return 0;
}
...................................................................................
  static inline void generic_handle_irq_desc(unsigned int irq, struct irq_desc *desc)
{
    desc->handle_irq(irq, desc); //调用流控层函数，对于gic的spi调用的是fast_eoi，在下文中的初始化中设置
}
```

二、系统启动时的中断初始化

```
DT_MACHINE_START(SOCFPGA, "Altera SOCFPGA")
    .smp            = smp_ops(socfpga_smp_ops),
    .map_io     = socfpga_map_io,
    .init_irq   = gic_init_irq,     /* 在start_kernel中的init_IRQ()中调用 */
    .handle_irq     = gic_handle_irq,   /* 在setup_arch中赋值给handle_arch_irq */
    .timer      = &dw_apb_timer,
    .nr_irqs        = SOCFPGA_NR_IRQS,
    .init_machine   = socfpga_cyclone5_init,
    .restart    = socfpga_cyclone5_restart,
```

```
    .reserve        = socfpga_ucosii_reserve,
    .dt_compat  = altera_dt_match,
MACHINE_END

#define DT_MACHINE_START(_name, _namestr)       \
static const struct machine_desc __mach_desc_##_name    \
 __used                         \
 __attribute__((__section__(".arch.info.init"))) = {    \
    .nr     = ~0,                   \
    .name       = _namestr,

..........................................................................
asmlinkage void __init start_kernel(void)
{
    ...
    setup_arch(&command_line);
    ...
    early_irq_init();
    init_IRQ();
    ...
}
```

```
void __init setup_arch(char **cmdline_p)
{
    struct machine_desc *mdesc;

    setup_processor();
    mdesc = setup_machine_fdt(__atags_pointer);     /* 获取机器描述符 */
    if (!mdesc)
        mdesc = setup_machine_tags(__atags_pointer, machine_arch_type);
    machine_desc = mdesc;
    machine_name = mdesc->name;

    setup_dma_zone(mdesc);

    if (mdesc->restart_mode)
        reboot_setup(&mdesc->restart_mode);

    init_mm.start_code = (unsigned long) _text;
    init_mm.end_code   = (unsigned long) _etext;
    init_mm.end_data   = (unsigned long) _edata;
    init_mm.brk    = (unsigned long) _end;

    /* populate cmd_line too for later use, preserving boot_command_line */
    strlcpy(cmd_line, boot_command_line, COMMAND_LINE_SIZE);
    *cmdline_p = cmd_line;

    parse_early_param();

    sort(&meminfo.bank, meminfo.nr_banks, sizeof(meminfo.bank[0]), meminfo_cmp, NULL);
    sanity_check_meminfo();
    arm_memblock_init(&meminfo, mdesc);
```

```c
    paging_init(mdesc);     /* 初始化一些页表映射和异常向量表 */
    request_standard_resources(mdesc);

    if (mdesc->restart)
        arm_pm_restart = mdesc->restart;    /*  */

    unflatten_device_tree();

#ifdef CONFIG_SMP
    if (is_smp()) {
        smp_set_ops(mdesc->smp);
        smp_init_cpus();
    }
#endif

    if (!is_smp())
        hyp_mode_check();

    reserve_crashkernel();

    tcm_init();

#ifdef CONFIG_MULTI_IRQ_HANDLER
    handle_arch_irq = mdesc->handle_irq;    /*  */
#endif

#ifdef CONFIG_VT
#if defined(CONFIG_VGA_CONSOLE)
    conswitchp = &vga_con;
#elif defined(CONFIG_DUMMY_CONSOLE)
    conswitchp = &dummy_con;
#endif
#endif

    if (mdesc->init_early)
        mdesc->init_early();
}
```

```c
int __init early_irq_init(void)
{
    int i, initcnt, node = first_online_node;
    struct irq_desc *desc;

    init_irq_default_affinity();

    /* Let arch update nr_irqs and return the nr of preallocated irqs */
    initcnt = arch_probe_nr_irqs();
    printk(KERN_INFO "NR_IRQS:%d nr_irqs:%d %d\n", NR_IRQS, nr_irqs, initcnt);

    if (WARN_ON(nr_irqs > IRQ_BITMAP_BITS))
        nr_irqs = IRQ_BITMAP_BITS;
```

```
    if (WARN_ON(initcnt > IRQ_BITMAP_BITS))
        initcnt = IRQ_BITMAP_BITS;

    if (initcnt > nr_irqs)
        nr_irqs = initcnt;

    for (i = 0; i < initcnt; i++) {
        desc = alloc_desc(i, node, NULL);
        set_bit(i, allocated_irqs);
        irq_insert_desc(i, desc);
    }
    return arch_early_irq_init();
}
```

三、中断映射

四、中断注册