practice07

2021011158 김선희

Kernel porting | Implement Software Interrupt Entering/Leaving Routine, Timer Interrupt

Software Interrupt

Code Implementation

HAL_arch_startup.S

```
/* vh_entering_swi */
vh_software_interrupt:
    /* Store the current sp value */
    str       sp, vk_save_swi_mode_stack_ptr

    /* Store lr, sp and general registers of previous mode on the stack */
    stmfd   sp, {r14}^
    sub     sp, sp, #4
    stmfd   sp, {r13}^
    sub     sp, sp, #4
    stmfd   sp!, {r0-r12}

    /* Store SPSR and lr on the stack */
    mrs     r0, spsr_all
    stmfd   sp!, {r0, lr}

    /* Disable IRQ */
    mrs     r0, cpsr
    orr     r0, r0, #0x80
    msr     cpsr, r0

    /* Store the current sp value */
    str       sp, vk_save_swi_current_tcb_bottom

    /* Store the parameters in the r0 register */
    ldr       r0, [sp, #8]

    /* Jump to SWI handler */
    bl     vk_swi_classifier

vh_leaving_swi:
    /* Restore all registers that were saved on the stack */
    ldmfd   sp!, {r0, lr}
    msr     spsr_cxsf, r0
    ldmfd   sp!, {r0-r12}
    ldmfd   sp, {r13}^
    add     sp, sp, #4
    ldmfd   sp, {r14}^
    add     sp, sp, #4

    /* Return to original routine */
    movs   pc, lr
```

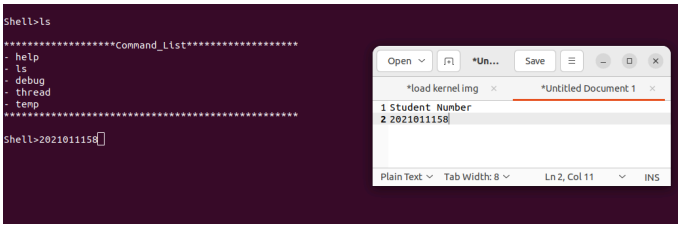
Most implementations are identical to hardware interrupt handling.

The difference is in step2(course 6) of the entering routine. Unlike in hardware interrupt handling, the IRQ must be disabled directly. So I added the following 3 lines of code to disable IRQ.

```
    mrs     r0, cpsr
    orr     r0, r0, #0x80
```

```
    msr     cpsr, r0
```

Result



Timer

Code Implementation

kernel_start.c

```
void set_interrupt(void)
{
    // disable  interrupts
    // * GICC_CTRL: Enables the signaling of interrupts
    GICC_CTRL &= 0;
    GICD_CTRL &= 0;

    // interrupt setting
    vh_serial_irq_enable();
    vh_timer_irq_enable();

    // set priority mask to the lowest level (to accept interrupts of any priority level)
    GICC_PMR = 0xff;

    // enable interrupts
    GICC_CTRL |= 1;
    GICD_CTRL |= 1;
}
```

timer.c

```
void vh_timer_irq_enable()
{
    // clear active & pending status
    GICD_ICACTIVER(INTERRUPT_ID_TIMER / 32) |= (1 << (INTERRUPT_ID_TIMER % 32));
    GICD_ICPENDR(INTERRUPT_ID_TIMER / 32) |= (1 << (INTERRUPT_ID_TIMER % 32));

    // enable interrupt
    GICD_ISENABLER(INTERRUPT_ID_TIMER / 32) |= (1 << (INTERRUPT_ID_TIMER % 32));

    // set interrupt target (to cpu 0)
    GICD_ITARGETSR(INTERRUPT_ID_TIMER / 4) |= (1 << ((INTERRUPT_ID_TIMER % 4) * 8));

}

void vh_timer_interrupt_handler(void)
{
    vk_timer_irq_disable();
    vh_save_thread_ctx(vk_timer_save_stk);
```

```
    // timer interrupt clear & enable
    vk_timer_irq_enable();

    vk_sched_save_tcb_ptr = (unsigned int)vk_timer_save_stk;
    vk_timer_flag = 1;

    ++(vk_current_thread->cpu_tick);
    if(vk_sched_lock==0) {
        vk_swi_scheduler();
    }
}

void vk_timer_irq_enable()
{
    write_cntp_tval(get_1sec() / 16);
    write_cntp_ctl(1);
}

void vk_timer_irq_disable(void)
{
    write_cntp_ctl(0);
}
```

Used the functions the same way we used them in practice06.

Result