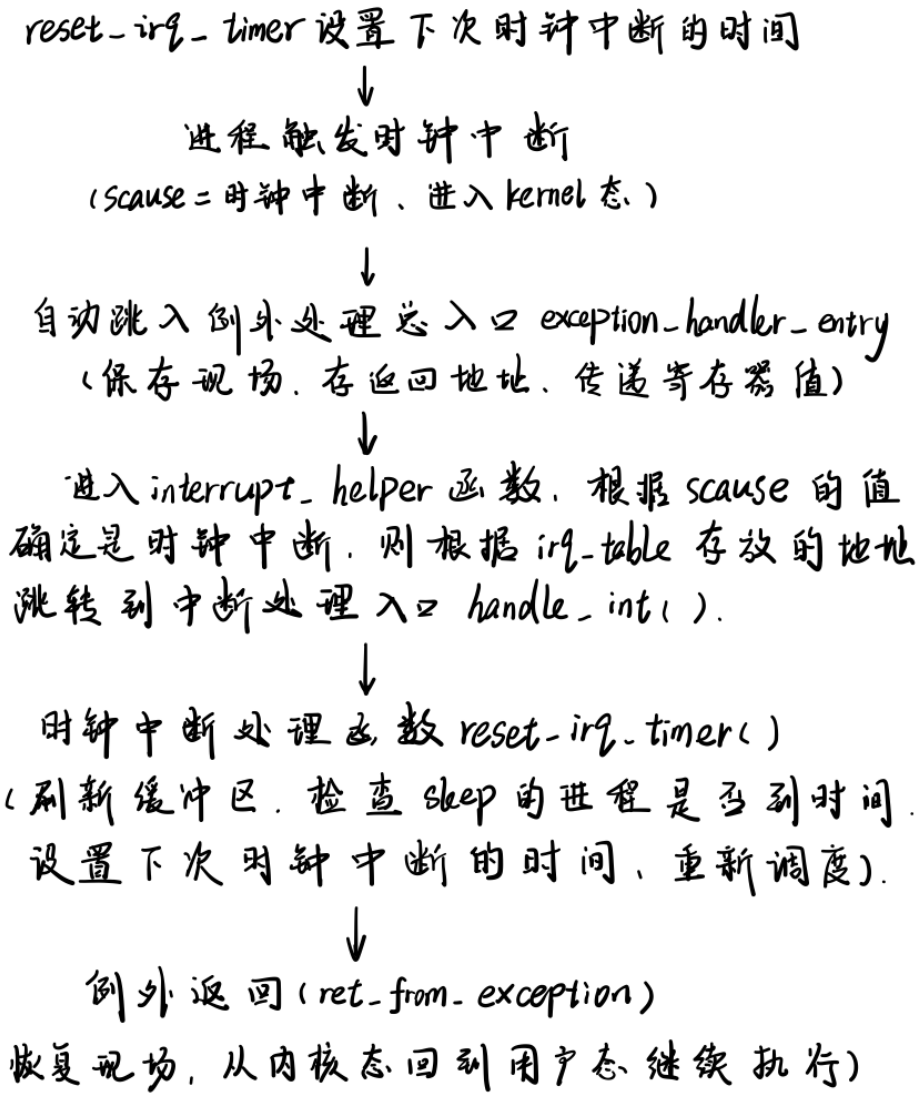
Project2 A Simple Kernel 设计文档（Part II）

中国科学院大学
[姓名] 蔡洛姗
[日期] 2020/11/8

1. 时钟中断、系统调用与 blocking sleep 设计流程

（1）时钟中断处理的流程：



（2）你所实现的时钟中断的处理流程中，何时唤醒 sleep 的任务？

在执行时钟中断处理函数 reset_irq_time 时，用 timer_check()函数检查 timer 队列中每一个 sleep 的任务是否到时间，如果到了时间就把该任务对应的 pcb 放入 ready_queue 中，等待执行。

（3）你实现的时钟中断处理流程和系统调用处理流程有什么相同步骤，有什么不同步

骤？

相同：都从例外处理总入口 exception_handler_entry 进入开始处理，从 ret_from_exception 返回用户态继续执行，需要保存和恢复现场。

不同：系统调用由任务主动触发，时钟中断由硬件被动触发。根据 scause 寄存器的值，如果是系统调用则跳到 handle_syscall 再根据系统调用好调用不同的系统调用处理函数，如果是时钟中断则跳到 handle_int 处理时钟中断。

（4）设计、实现或调试过程中遇到的问题和得到的经验

在调试过程中遇到的最大的问题是关于栈指针的设置。一开始由于对用户态和核心态的理解不够充分，直接把任务运行在了内核栈上，没有在汇编代码中实现栈指针的保存与转换。这导致了保存在内核栈的用户态变量在系统调用或时钟中断陷入内核之后就会被冲掉，最开始的变量就是 printlocation，因此我的终端只打印了一行。考虑到这一点之后，在保存和恢复现场的过程中更需要小心注意 sp 和 tp 的维护。由于需要用 sp 来作为保存恢复的基址，使用前后都需要注意 sp 的值的恢复；tp 保存的是 current_running 的值，也就是当前任务的地址，我一开始使用了 la 来赋值，其实是又取了 current_running 的地址，因此出现了错误。

2. 基于优先级的调度器设计（做 C-Core 的同学请回答）

（1）你实现的调度策略中，优先级是怎么定义的，测试用例中有几个任务，各自优先级是多少，结果如何体现优先级的差别？

我在初始化的时候为每个任务分配了一个优先级，且将优先级的值放入 pcb 结构体的 list 结构中；在调度过程中会按照优先级的高低将 pcb 放入 ready_queue 中，这样就绪队列就是一个优先级高的在队头，优先级低的在队尾的有序队列了；因此出队列时会优先选择优先级高的任务执行；为了避免优先级低的任务饿死，每次执行完一个任务并将其重新放入就绪队列时就会把它的优先级减 1，这样原本优先级高的任务会逐渐降低优先级，让给其它任务执行。

3. Context-switch 开销测量的设计思路（做 C-Core 的同学请回答）

（1） 你的测试用例和结果介绍

我选择在 reset_irq_timer 调用 do_scheduler 前后分别记录其时钟周期数，然后相减得到完成一次调度所用周期数，考虑到调度前后位于两个不同的进程，因此如果是局部变量就会在回收栈时被冲掉，因此需要用全局变量来保存前后测得的时钟周期数。

考虑到一次调度的时间较小，所以我直接打印了周期数而没有除以 time_base 计算时间，测试得到的时间大约是 1000~2000ticks。

4. 关键函数功能

```
//实现时钟中断
void reset_irq_timer()
{
    // TODO clock interrupt handler.
    screen_reflush();
    timer_check();
    uint64_t stime_value;
    stime_value = get_ticks() + time_base/200;
```

```
    sbi_set_timer(stime_value);
    // note: use sbi_set_timer
    // remember to reschedule
    sched_begin_tick = get_ticks();
    do_scheduler();
    sched_end_tick = get_ticks();
    sched_ticks = sched_end_tick-sched_begin_tick;
        vt100_move_cursor(1, 9);
        printk("do_scheduler costs (%ld) ticks.\n\r",sched_ticks);
}

//task 3：根据 scause 确定例外种类，调用不同的例外处理函数进行处理
void interrupt_helper(regs_context_t *regs, uint64_t stval, uint64_t cause)
{
    // TODO interrupt handler.
    // call corresponding handler by the value of `cause`
    if(cause < 0x8000000000000000){
        exc_table[regs->scause](regs,0,cause);
    }else{
        irq_table[regs->scause-0x8000000000000000](regs,1,cause);
    }
}

void handle_int(regs_context_t *regs, uint64_t interrupt, uint64_t cause)
{
    reset_irq_timer();
}

//task 4
void init_exception()
{
    /* TODO: initialize irq_table and exc_table */
    /* note: handle_int, handle_syscall, handle_other, etc.*/
    int i;
    for ( i = 0; i < IRQC_COUNT; i++)
        irq_table[i] = &handle_other;
    irq_table[IRQC_S_TIMER] = &handle_int;
    for ( i = 0; i < EXCC_COUNT; i++)
        exc_table[i] = &handle_other;
    exc_table[EXCC_SYSCALL] = &handle_syscall;
    setup_exception(); //part 1 don't need
}
```

```
//优先调度
void do_scheduler(void)
{
    __asm__ __volatile__("csrr x0, sscratch\n");
    pcb_t *prev_running;
    prev_running = current_running;
    /* priority scheduler */
    if(current_running->status!=TASK_BLOCKED){
        current_running->status=TASK_READY;
        if(current_running->pid!=0){
            current_running->list.priority--;
            list_add_priority(&current_running->list,&ready_queue);
        }
    }

    if(!list_empty(&ready_queue)){
        current_running = list_entry(ready_queue.prev, pcb_t, list);
        list_del(ready_queue.prev);
    }
    current_running->status=TASK_RUNNING;
    switch_to(prev_running, current_running);
}

//保存上下文
.macro SAVE_CONTEXT
  .local _restore_kernel_tpsp
  .local _save_context

  csrrw tp, CSR_SSCRATCH, tp
  bnez tp, _save_context

_restore_kernel_tpsp:
  csrr tp, CSR_SSCRATCH
  sd sp, PCB_KERNEL_SP(tp)
_save_context:
  sd sp, PCB_USER_SP(tp)
  ld sp, PCB_KERNEL_SP(tp)
  addi sp, sp, -(OFFSET_SIZE)

  sd  zero,OFFSET_REG_ZERO(sp)
  sd  ra  ,OFFSET_REG_RA  (sp)
  ......
  ......
  sd  t6  ,OFFSET_REG_T6  (sp)
```

```
  ld t0, PCB_USER_SP(tp)
  sd t0  ,OFFSET_REG_SP  (sp)

  li t0, SR_SUM | SR_FS
  csrr t0 ,CSR_SSTATUS
  sd   t0 ,OFFSET_REG_SSTATUS (sp)
  csrr t0 ,CSR_SEPC
  sd   t0 ,OFFSET_REG_SEPC    (sp)
  csrr t0 ,CSR_STVAL
  sd   t0 ,OFFSET_REG_SBADADDR(sp)
  csrr t0 ,CSR_SCAUSE
  sd   t0 ,OFFSET_REG_SCAUSE  (sp)
.endm

//恢复上下文
.macro RESTORE_CONTEXT
  /* TODO: Restore all registers and sepc,sstatus */
  //sp have been set in switch_to
  addi t0, sp, OFFSET_SIZE
  sd  t0, PCB_KERNEL_SP   (tp)

  ld   t0 ,OFFSET_REG_SSTATUS (sp)
  csrw CSR_SSTATUS, t0
  ld   t0 ,OFFSET_REG_SEPC    (sp)
  csrw CSR_SEPC   , t0

  ld  zero,OFFSET_REG_ZERO(sp)
  ......
  ......
  ld  t6  ,OFFSET_REG_T6  (sp)

  ld  tp  ,OFFSET_REG_TP  (sp)
  ld  sp  ,OFFSET_REG_SP  (sp)
.endm
```