操作系统实验报告Prj2-Part2

万炎广 2017K8009907017

主要设计思路

例外处理

中断处理

首先在代码 exception_handler_entry 中，写入例外处理的第二级； 这里的程序等下会在初始化的时候，通过memcpy拷贝到内存的0x80000180处。 由于发生例外的时候硬件作为第一级处理，会自动跳转到0x80000180处执行； 在0x80000180处进行处理流程如下：

1. 先进行关中断。（将status最后一位设置为0）
   这么做的目的是不让系统在核心态进行处理的时候，被例外中断。
2. 进行现场的保存。
   无论是系统调用例外，还是中断，现场都需要得到保存，即使接下来运行的进程仍然是现在的进程。
3. 进行例外原因判断。
   具体做法是取CP0_CAUSE的6-2位置，根据引发例外的原因，分别跳转到各个例外处理的程序入口。

在跳转了之后，首先看中断处理程序： 中断处理程序的大体流程如下：

1. 获取CP0_STATUS和CPO_CAUSE，并作为interrupt_helper(中断帮手)的输入。
   在中断帮手里面，会完成进程调度（修改current_running）以及其他操作。
2. 恢复current_running对应进程的现场。
3. 由于是时钟中断，因此要清除CP0_COUNT值。
4. 重置COMPARE以清中断。（否则开中断后马上迎来另一次中断）
5. 使用eret跳转回到EPC中断前程序位置。

由于中断处理程序使用EPC作为跳转目标，因此在初始化的过程中，可以选取EPC存放测试进程入口。

系统调用

系统表用的方法和中断处理比较类似，但是有些细节不同。

1. 首先，保存现场得到的EPC应当+4，因为发生例外时会认为该条指令未被执行；
   而不进行+4操作的话，就会不断触发syscall而陷入死循环。
2. 系统调用的时候不需要对于count寄存器进行清除，但是要保证进行eret跳转回去之前，COUNT值是小于COMPARE值的。

系统处理程序的大体流程如下：

1. 将v0,a0,a1,a2的四个寄存器的值转为：a0,a1,a2,a3作为输入。v0是系统调用的规范，表示要完成什么处理。
2. 跳转到system_call_helper，进一步决定跳转到具体的系统处理函数进行处理。
3. 恢复现场并使用eret回到系统调用触发位置。（EPC要+4）

调度设计

采用的调度设计是进行优先级的调度设计。具体思路如下：

首先在初始化的时候，会设定每一个进程的优先级。并且时刻（进程上一次的运行的时刻，初始为0）记录下来。
在进行调度的时候，将当前时刻减去进程上一次运行的时刻（等待时间）加上设定的优先级，作为最终判断依据。
选取最高的优先级进行调度运行。

这样做的一个好处是所有的程序都可以得到运行，并且不需要重复设定其优先级。 坏处是一旦优先级被设定就很难修改。（要设计新的syscall）

唤醒睡眠程序

在每一次进行调度 `scheduler()` 的时候，对于 `sleep_queue` 进行一次轮询，如果当前睡眠时间已经完成，则将其唤醒进入准备队列中。

bonus设计

bonus的题目为设计一个程序多把锁。 具体设计如下：

一个程序多把锁的目的是当一个程序需要使用到多个文件等情况会出现。具体设计其实和单把锁类似。
首先要准备多把锁，以及多个阻塞队列。每一把锁对应每一个阻塞队列。
1.由程序连续申请两把锁（这里以两把锁作为示范）
2.如果申请锁的时候受到了阻塞，则会进入相应锁的阻塞队列。等待锁被释放之后再进行操作。
3.解锁时程序连续释放两把锁。

之所以要进行连续地申请和连续地解锁，为了防止出现死锁情况：
程序A已获得锁1，申请锁2被阻塞。
程序B已获得锁2，申请锁1被阻塞。

注意事项：

时钟中断要清除CP0_COUNT值。
时钟中断中重置COMPARE以清中断。（否则开中断后马上迎来另一次中断）
系统调用的保存现场得到的EPC应当+4。
注意保存screen_cursor的值，否则会导致打印的光标乱飘。

附录：具体代码：

Exception_handler_entry

```
NESTED(exception_handler_entry, 0, sp)
exception_handler_begin:
    mfc0    k0, CP0_EPC
    nop
    mfc0    k0, CP0_STATUS
    nop
    li      k1, 0xfffffffe
    and     k0, k0, k1
    mtc0    k0, CP0_STATUS
    nop

testsave:

    SAVE_CONTEXT(USER)
    li      sp, 0xa0f05000

    mfc0    k0, CP0_CAUSE
    nop
    andi    k0, 0x7c
    la      k1, exception_handler
```

```
    add     k0, k0, k1
    lw      k0, 0(k0)
    jr      k0
    nop
    nop
    //TODO close interrupt
    // jmp exception_handler[i] which decided by CP0_CAUSE
    // Leve2 exception Handler.
exception_handler_end:
END(exception_handler_entry)
```

中断处理

```
NESTED(handle_int, 0, sp)
    mfc0    a0, CP0_STATUS
    mfc0    a1, CP0_CAUSE
    nop
    addi    sp, sp, -8            # sp = sp + -8

    jal        interrupt_helper               # jump to interrupt_helper
    addi    sp, sp, 8
    nop
    RESTORE_CONTEXT(USER)
    li      k0, INT_TIME
    mtc0    zero, CP0_COUNT
    nop
    mtc0    k0, CP0_COMPARE
    nop
    mfc0    k1, CP0_STATUS
    nop
    li      k0, 0x1
    or      k1, k1, k0
    mtc0    k1, CP0_STATUS
    nop
    eret
    nop

    // interrupt handler
    // Leve3 exception Handler.
END(handle_int)
```

调度

```
void scheduler(void)
{
    if(!queue_is_empty(&sleep_queue)){
        check_sleeping();
    }
    current_running->cursor_x = screen_cursor_x;
    current_running->cursor_y = screen_cursor_y;
    if(current_running->status == TASK_RUNNING || current_running->status == TASK_READY) {
        current_running->status = TASK_READY;
        queue_push(&ready_queue, current_running);
    }

    int maxpriority = -1;
    int temptime = get_timer();
    int tempprio;
    int i =1;

    for(; i<= 15;i++) {
        if(pcb[i].status !=TASK_READY) continue;
```

```
        tempprio = prio[i][0] + temptime - prio[i][1];
        if(tempprio > maxpriority) {
            maxpriority = tempprio;
            current_running = &pcb[i];
        }
    }
    prio[current_running->pid - 1][1]= temptime;

    //current_running = &pcb[7];

    //printk("test\n");
    queue_remove(&ready_queue, current_running);
    current_running->status = TASK_RUNNING;

    screen_cursor_x = current_running->cursor_x;
    screen_cursor_y = current_running->cursor_y;
}
```

系统调用

```
NESTED(handle_syscall, 0, sp)
    addi    sp, sp, -16
    move    a3, a2
    move    a2, a1
    move    a1, a0
    move    a0, v0
    jal        system_call_helper
    nop

    addi    sp, sp, 16

    RESTORE_CONTEXT(USER)
    li      k0, INT_TIME
    mtc0    k0, CP0_COMPARE
    nop

    mfc0    k0, CP0_COUNT
    nop
    li      k1, INT_TIME
    addi    k1, k1, -100
    slt     k0, k0, k1

    bnez    k0, OKTOGO
    mtc0    k1, CP0_COUNT
    nop

OKTOGO:
    mfc0    k0, CP0_STATUS
    li      k1, 0x00000001
    or      k0, k0, k1
    mtc0    k0, CP0_STATUS
    nop
    eret
    // system call handler
END(handle_syscall)
```