Lab 2

4.1 准备工程

- 修改对于 puti、putc、print.c、print.h 的引用（略）
- 依葫芦画瓢修改 head.S 和 vmlinux.lds （略）

4.2 开启异常处理

STIE为bit[6]，用0x20置位；SIE为bit[2]，用0x2置位。为了不影响到寄存器中其他位上的信息，用or

命令进行操作。

第一次设置时间中断，手动设置a0（传入的参数，即要设置的mtimecmp的值）、a6（ExtensionID = 0）、a7（FunctionID = 0）的参数调用ecall来设置mtimecmp。

```
.extern start_kernel

    .section .text.init
    .globl _start
_start:
        # set stvec = _traps
    la t0, _traps
    csrw stvec, t0

        # set sie[STIE] = 1
    csrr t0, sie
    ori t0, t0, 0x20
    csrw sie, t0

        # set first time interrupt
    rdtime t0
    li t1, 10000000
    add a0, t0, t1
    li a6, 0
    li a7, 0
    ecall

        # set sstatus[SIE] = 1
    csrr t0, sstatus
    ori t0, t0, 0x2
    csrw sstatus, t0

    la sp, boot_stack_top
    call start_kernel

    .section .bss.stack
    .globl boot_stack
boot_stack:
    .space 4096

    .globl boot_stack_top
boot_stack_top:
```

4.3 实现上下文切换

将除了x0以外的寄存器以及sepc都存入栈中（因为x0始终为0，所以不用管），移动栈的指针到此时的栈顶；用a0和a1传递参数，跳转到trap_handler处理中断。处理完毕后再恢复所有寄存器的值，然后sret（S态下从中断返回）。

```
    .section .text.entry
    .align 2
    .globl _traps
_traps:

        # 1. save 32 registers and sepc to stack
        sd ra, -8(sp)
        sd sp, -16(sp)
        sd gp, -24(sp)
        sd tp, -32(sp)
        sd t0, -40(sp)
        sd t1, -48(sp)
        sd t2, -56(sp)
        sd s0, -64(sp)
        sd s1, -72(sp)
        sd a0, -80(sp)
        sd a1, -88(sp)
        sd a2, -96(sp)
        sd a3, -104(sp)
        sd a4, -112(sp)
        sd a5, -120(sp)
        sd a6, -128(sp)
        sd a7, -136(sp)
        sd s2, -144(sp)
        sd s3, -152(sp)
        sd s4, -160(sp)
        sd s5, -168(sp)
        sd s6, -176(sp)
        sd s7, -184(sp)
        sd s8, -192(sp)
        sd s9, -200(sp)
        sd s10, -208(sp)
        sd s11, -216(sp)
        sd t3, -224(sp)
        sd t4, -232(sp)
        sd t5, -240(sp)
        sd t6, -248(sp)
        csrr t0, sepc
        sd t0, -256(sp)
        addi sp, sp, -256


    # -----------

        # 2. call trap_handler，a0和a1分别传递scause和sepc的参数
        csrr a0, scause
        csrr a1, sepc
        call trap_handler

    # -----------

```

```
    # 3. restore sepc and 32 registers (x2(sp) should be restore last) from stack

    ld t0, 0(sp)
    csrw sepc, t0
    ld t6, 8(sp)
    ld t5, 16(sp)
    ld t4, 24(sp)
    ld t3, 32(sp)
    ld s11, 40(sp)
    ld s10, 48(sp)
    ld s9, 56(sp)
    ld s8, 64(sp)
    ld s7, 72(sp)
    ld s6, 80(sp)
    ld s5, 88(sp)
    ld s4, 96(sp)
    ld s3, 104(sp)
    ld s2, 112(sp)
    ld a7, 120(sp)
    ld a6, 128(sp)
    ld a5, 136(sp)
    ld a4, 144(sp)
    ld a3, 152(sp)
    ld a2, 160(sp)
    ld a1, 168(sp)
    ld a0, 176(sp)
    ld s1, 184(sp)
    ld s0, 192(sp)
    ld t2, 200(sp)
    ld t1, 208(sp)
    ld t0, 216(sp)
    ld tp, 224(sp)
    ld gp, 232(sp)
    ld ra, 248(sp)
    ld sp, 240(sp)

# -----------

sret    # 4. return from trap

# -----------

```

4.4 实现异常处理函数

scause中，bit[XLEN]=1时表明这是个interrupt；其中S态的时钟中断代码为5。因此要判断scause的第一位是1和后四位是0101。

```c
// trap.c

#include "printk.h"
#include "clock.h"

void trap_handler(unsigned long scause,unsigned long sepc){
  if(scause >> 63 == 1 && (scause << 1) == 10){ //判断第一位是1和后四位是0101

      printk("[S] Supervisor Mode Timer Interrupt\n");//打印中断信息
      clock_set_next_event();//设置下一次时钟中断

  }
}
```

4.5 实现时钟中断相关函数

```c
// clock.c

// QEMU中时钟的频率是10MHz, 也就是1秒钟相当于10000000个时钟周期。
unsigned long TIMECLOCK = 10000000;

unsigned long get_cycles() {
  // 使用 rdtime 编写内联汇编，获取 time 寄存器中 (也就是mtime 寄存器 )的值并返回
  unsigned long time = 0;
    asm volatile (
    "rdtime %[time]\n"
    : [time] "=r" (time)
    :
    : "memory"
  );
  return time;

}

void clock_set_next_event() {
  // 下一次 时钟中断 的时间点
  unsigned long next = get_cycles() + TIMECLOCK;

  // 使用 sbi_ecall 来完成对下一次时钟中断的设置，functionID为0x0
  sbi_ecall(0x0, 0x0, next, 0, 0, 0, 0, 0);
}


```

4.6 修改Makefile、编译并测试

根据新加的.c文件加入对应的.h头文件（在lab2/arch/riscv/include中加入了一个clock.h），由于Makefile是自动寻找.c对应的.h文件的，所以没啥好改的。

clock.h

```c
#pragma once

unsigned long get_cycles();
void clock_set_next_event();
```

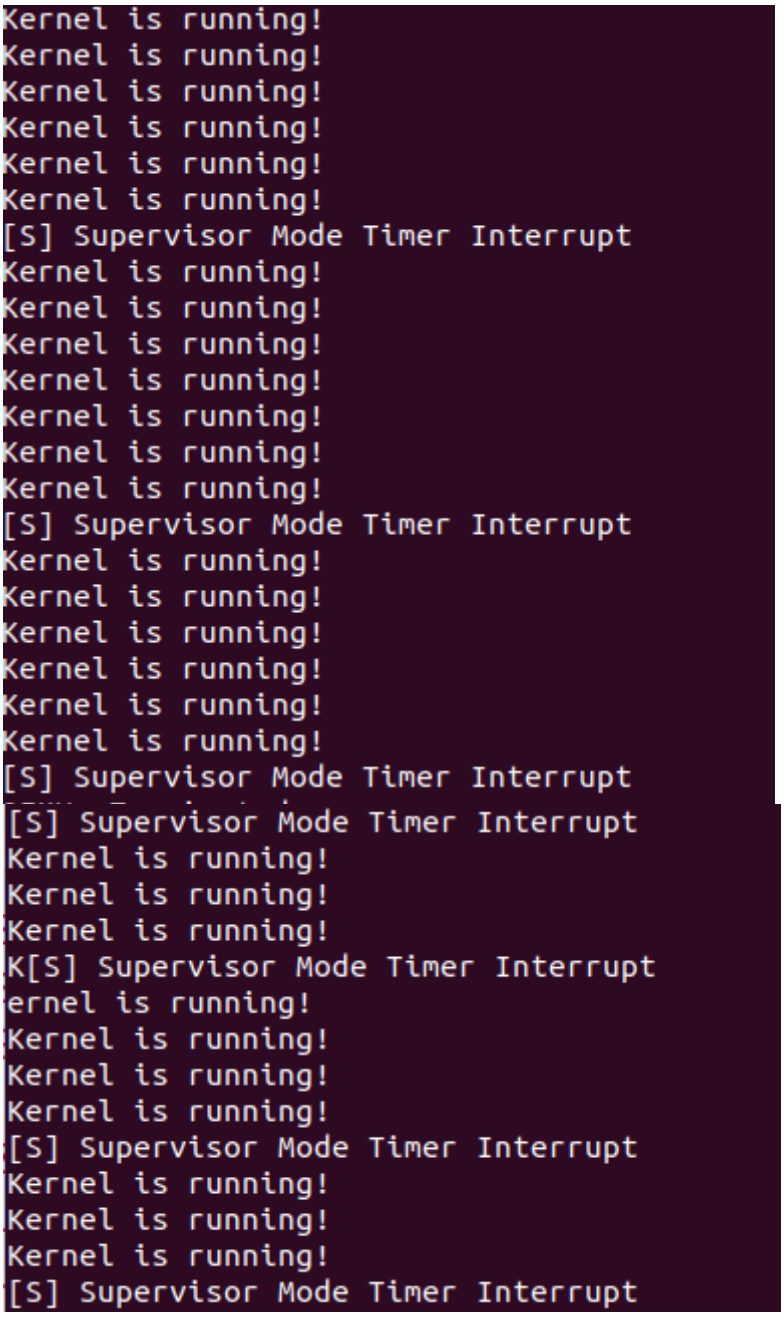
test.c

`cnt`是为了控制内核正常运行时候的语句的输出速率。（但是控制得似乎不太稳定，基本情况是打印2～3句会有一次时钟中断提示，但也有6～7次打印后才输出提示的情况）

```
unsigned long cnt = 100000000;

void test() {
  while (1) {

  cnt--;
  if (cnt==0) {
       printk("Kernel is running!\n");
    cnt = 100000000;
  }
  }
}
```



4.7 思考题

4.7.1 解释 `MIDELEG` 的含义

0x0000000000000222是把bit[1]、bit[5]、bit[9]置为1，表明将三种中断（software, timer, external）交给S模式处理。