实验 5—RV64 时钟中断处理

姓名：张云策 学号：3200105787 学院：计算机科学与技术学院

课程名称：计算机系统 II 同组学生姓名：/

实验时间：2021. 实验地点：紫金港机房 指导老师：申文博

一、 实验目的和要求

- 学习 RISC-V 的异常处理相关寄存器与指令，完成对异常处理的初始化。
- 理解 CPU 上下文切换机制，并正确实现上下文切换功能。
- 编写异常处理函数，完成对特定异常的处理。
- 调用 OpenSBI 提供的接口，完成对时钟中断事件的设置。

二、 实验内容和原理

2.1 实验内容

说明：参照“实验目的”

2.2 设计模块

说明：

- 修改 vmlinux.lds 文件，使得中断处理可以正常工作
- 修改 head.S 文件，使得_start 嵌入启动部分，可以进行中断处理，并且对 stvec 设置为写入地址
- 修改 entry.S 文件，使得 scause 以及 sepc 的值可以传入异常处理模块中
- 建立 trap.C 文件，使得 kernel 具有异常处理功能（实现时钟中断）
- Clock.C 文件，使异常处理模块具有获取 time 寄存器值及设置时钟中断事件的能力
- Make file，对于编译过程进行一些修改。

三、 主要仪器设备

Docker in lab3

四、 操作方法与实验步骤

4.1 操作方法

说明：参照 lab4 进行 kernel 的编译。

4.2 实验代码

Vmlinux.lds

```
BASE_ADDR = 0x80200000;
SECTIONS
{
    . = BASE_ADDR;
    .text : {
        *(.text.init)
        *(.text.entry)
        *(.text)
        *(.text.*)
     }
    .rodata : {
        *(.rodata)
        *(.rodata.*)
    }
    .data : {
        *(.data)
        *(.data.*)
    }
    bss_start = .;
    .bss : {
        *(.sbss)
        *(.sbss.*)
        *(.bss)
        *(.bss.*)
    }
    bss_end = .;
    . += 0x8000;
    stack_top = .;
    _end = .;
}

```

Head.S

```
.extern start_kernel
.extern _start_a
.extern stack_top
    .section .text.init
    .globl _start
_start:
    la sp,stack_top
    call _start_a
.section .bss.stack
    .globl boot_stack
boot_stack:
    .space 4096

    .globl boot_stack_top
boot_stack_top:
```

_start_a.c

```c
#include "riscv.h"

extern void clock_init(void);

void intr_enable(void)
{
    unsigned long res = set_csr(sstatus, 2);
    unsigned long ss = read_csr(sstatus);
}

void intr_disable(void)
{
    unsigned long res = clear_csr(sstatus, 2);
}

void idt_init(void)
{
    extern void trap_s(void);
    write_csr(stvec, &trap_s);
    unsigned long st = read_csr(stvec);
}

void _start_a(void)
{
    idt_init();
    intr_enable();
    clock_init();
}
```

Trap.c

```c
#include "defs.h"

extern main(), puts(), put_num(), ticks;
extern void clock_set_next_event(void);

void trap_handler(unsigned long scause, unsigned long spec)
{
    // interrupt
    if (cause & 0x8000000000000000)
    {
        // supervisor timer interrupt
        if (cause == 0x8000000000000005)
        {
            clock_set_next_event();
            puts("[S] Supervisor Mode Timer Interrupt ");
            ++ticks;
            puts("\n");
        }
    }
}
```

Entry.S

```
.section .text.entry
.align 2
.global _traps
.extern trap_handler
.equ reg_size, 0x8

_traps:
    #save callee-saved registers and spec
    addi sp,sp, -144
    sd ra,136(sp)
    sd s0,128(sp)
    addi s0,sp,144
    sd t0,-24(s0)
    sd t1,-32(s0)
    sd t2,-40(s0)
    sd a0,-48(s0)
    sd a1,-56(s0)
    sd a2,-64(s0)
    sd a3,-72(s0)
    sd a4,-80(s0)
    sd a5,-88(s0)
    sd a6,-96(s0)
    sd a7,-104(s0)
    sd t3,-112(s0)
    sd t4,-120(s0)
    sd t5,-128(s0)
    sd t6,-132(s0)
    csrr t3,sepc
    sd t3,-140(s0)
    csrr a0,scause
    call trap_handler
    # call trap_handler(scause)
    # load sepc and callee-saved registers
    ld t3,-140(s0)
    csrw sepc,t3
    ld t0,-24(s0)
    ld t1,-32(s0)
    ld t2,-40(s0)
    ld a0,-48(s0)
    ld a1,-56(s0)
    ld a2,-64(s0)
    ld a3,-72(s0)
    ld a4,-80(s0)
    ld a5,-88(s0)
    ld a6,-96(s0)
    ld a7,-104(s0)
    ld t3,-112(s0)
    ld t4,-120(s0)
    ld t5,-128(s0)
    ld t6,-132(s0)
    ld s0,128(sp)
    ld ra,136(sp)
    addi sp,sp,144
    sret
```

Clock.c

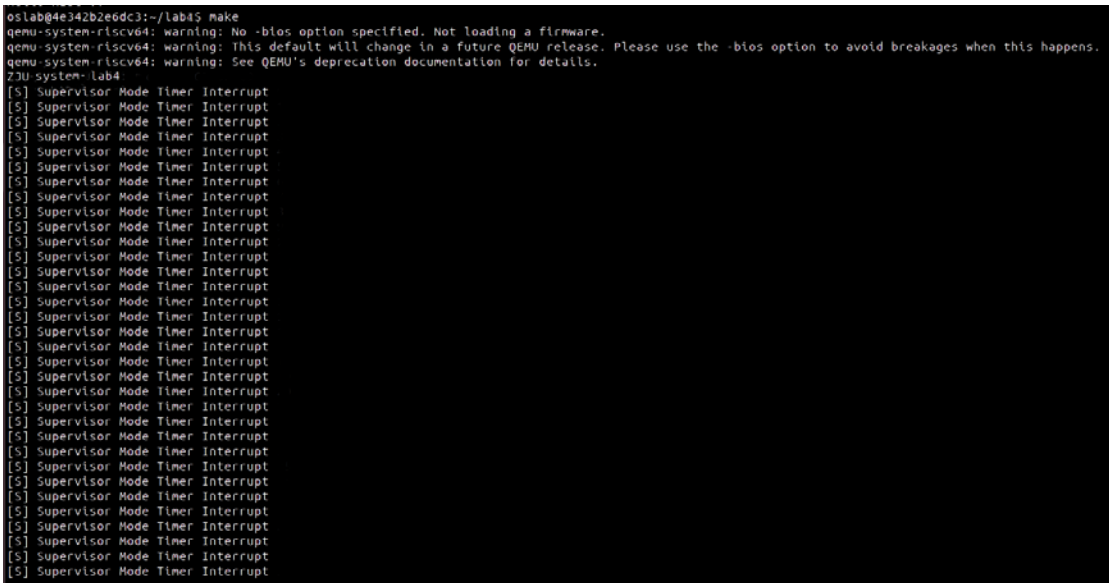
```c
//clock.c
#include "defs.h"
#include "riscv.h"
volatile unsigned long long ticks;
static uint64_t TIMECLOCK = 10000000;

const uint64_t SBI_SET_TIMER = 0;

extern sbi_call();
uint64_t get_cycles(void)
{
#if __riscv_xlen == 64
    uint64_t n;
    __asm__ __volatile__("rdtime %0"
                         : "=r"(n));
    return n;
#else
    uint32_t lo, hi, tmp;
    __asm__ __volatile__(
        "1:\n"
        "rdtimeh %0\n"
        "rdtime %1\n"
        "rdtimeh %2\n"
        "bne %0, %2, 1b"
        : "=&r"(hi), "=&r"(lo), "=&r"(tmp));
    return ((uint64_t)hi << 32) | lo;
#endif
}

void clock_set_next_event(void)
{
        puts("LAB5\n");
    __asm__ __volatile__(
        "li t1,32\n"
        "csrrs x0,sie,t1\n" ::
            :);

    unsigned long cs = read_csr(sie);

    uint64_t nextTime = get_cycles() + TIMECLOCK;
    trigger_time_interrupt(nextTime);
    uint64_t nextTime = get_cycles() + TIMECLOCK;
    trigger_time_interrupt(nextTime);
}

```

Makefile

```
export
CROSS_=riscv64-unknown-elf-
CC=${CROSS_}gcc
LD=${CROSS_}ld
OBJCOPY=${CROSS_}objcopy

DEBUG ?= true

ISA=rv64imafd
ABI=lp64

INCLUDE = -I$(shell pwd)/include -I$(shell pwd)/arch/riscv/include
CF = -g -march=$(ISA) -mabi=$(ABI) -mcmodel=medany -ffunction-sections -fdata-sections -nosta

# TASK_MM = -DPRIORITY
#TASK_MM = -DSJF

CFLAG = ${CF} ${INCLUDE} ${TASK_MM}

ifneq ($(DEBUG), )
CFLAG += -DDEBUG_LOG
endif

all: vmlinux

.PHONY: vmlinux run debug clean
vmlinux:
    # ${MAKE} -C init all
    # ${MAKE} -C lib all
    # ${MAKE} -C driver all
    ${MAKE} -C arch/riscv all
    ${LD} -T arch/riscv/kernel/vmlinux.lds arch/riscv/kernel/*.o -o vmlinux
    # init/*.o lib/*.o driver/*.o -o vmlinux
    $(shell test -d arch/riscv/boot || mkdir -p arch/riscv/boot)
    ${OBJCOPY} -O binary vmlinux arch/riscv/boot/Image
    nm vmlinux > System.map

run: vmlinux
    @qemu-system-riscv64 -nographic -machine virt -bios default -device loader,file=vmlinux,a

debug: vmlinux
    # @qemu-system-riscv64 -nographic -machine virt -kernel vmlinux -S -s -D log
    @qemu-system-riscv64 -nographic -machine virt -bios default -device loader,file=vmlinux,a
clean:
    # ${MAKE} -C init clean
    # ${MAKE} -C lib clean
    ${MAKE} -C arch/riscv clean
    $(shell test -f vmlinux && rm vmlinux)
    $(shell test -f System.map && rm System.map)
```

五、 实验结果与分析

由于是套改 lab4 文件，故显示 lab4xxx



六、 讨论、心得

说明：在设计实验过程中，对于 head.S 的设计有着较大误解，所以就求助了 TA，对于——start 部分进行了重写，并且对于处理同步异常时，对于返回地址的处理，也是很难的问题。

思考题：

MIDELEG 寄存器，可以实现 RISC v 的中断托管机制，从而使中断可以交由 Supervisor 模式处理，其中，bit[1]决定能否把 Delegate Supervisor Software Interrupt 实现，bit[5]决定能否把 Delegate Supervisor Timer Interrupt 实现，而 bit[9]决定能否把 Delegate Supervisor External Interrupt 实现，而其他位的值为留存下的中断地址等等。