浙江大学

本科实验报告

课程名称	操作系统
姓 名	
学 院	计算机科学与技术学院
系	计算机科学与技术系
专 业	计算机科学与技术
学 号	
指导教师	夏莹杰

2022 年 11 月 3 日

浙江大学操作系统实验报告

实验名称：**Lab 2: RV64 时钟中断处理**

电子邮件地址：　　手机：

实验地点：曹西503机房　　实验日期： 2022 年 11 月 3 日

一、实验目的和要求

- 学习 RISC-V 的 trap 处理相关寄存器与指令，完成对 trap 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 trap 处理函数，完成对特定 trap 的处理。
- 调用 OpenSBI 提供的接口，完成对时钟中断事件的设置。

二、实验过程

1、准备工程

- 此次实验基于 lab1所实现的代码进行。
- 在 lab1 中我们实现的 putiputs 使用起来较为繁琐，因此lab2提供了简化版的 printk。 **需要将之前所有 print.h puti puts 的引用修改为 printk.h printk。**
- 修改test.C，循环输出"kernel is running!\n"

```
#include "printk.h"
#include "defs.h"

// Please do not modify

void test() {
    int i = 0;
    while (1)
    {
        if (++i % 40000000 == 0)
        {
            i = 0;
            printk("kernel is running!\n");
        }
    }
}
```

2、开启 trap 处理

在运行 start_kernel 之前，我们要对上面提到的 CSR 进行初始化，初始化包括以下几个步骤：

1. 设置 stvec， 将 _traps (_traps 在 4.3 中实现) 所表示的地址写入 stvec，这里我们采用 Direct 模式, 而 _traps 则是 trap 处理入口函数的基地址。
2. 开启时钟中断，将 sie[STIE] 置 1。**STIE是第五位，与0x20(32)做or运算。**
3. 设置第一次时钟中断，即 sbi_set_timer，即sbi_ecall（0， 00， ...），其中**eid和fid分别对应a7和a6寄存器，a0寄存中存放参数arg0，代表10000000个时钟周期，即1秒。**
4. 开启 S 态下的中断响应， 将 sstatus[SIE] 置 1。

```
.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, 32
csrw sie, t0
# set first time interrupt
mv a6, x0
mv a7, x0
rdtime t0
la t1, 10000000
add a0, t0, t1
ecall
# set sstatus[SIE] = 1
csrr t0, sstatus
ori t0, t0, 2
csrw sstatus, t0
# ------------------
la sp, boot_stack_top
j start_kernel
```

```
27
28 .section .bss.stack
29 .globl boot_stack
30 boot_stack:
31 .space 4096 * 4 # <-- change to your stack size
32
33 .globl boot_stack_top
34 boot_stack_top:
```

3、实现上下文切换

我们要使用汇编实现上下文切换机制， 包含以下几个步骤：

1. 在 arch/riscv/kernel/ 目录下添加 entry.S 文件。
2. 保存 CPU 的寄存器（上下文）到内存中（栈上）：
    1. 需要先**开辟栈**，然后保存通用寄存器的值到开辟的栈上去。
    2. **sp需要保存到sscratch中**，x0不必保存。
3. 将 scause 和 sepc 中的值传入 trap 处理函数 trap_handler ， 即分别保存到**a0，a1**两个函数参数寄存器中。
4. 在完成对 trap 的处理之后， 我们从内存中（栈上）恢复CPU的寄存器（上下文）。
5. 从 trap 中返回。

```
1 .section .text.entry
2 .align 2
3 .globl _traps
4 _traps:
5 csrw sscratch, sp
6 # 1. save 32 registers and sepc to stack
7 # addi sp, sp, -32*8
8 # sd registers
9 ...
10 # 2. call trap_handler
11 csrr a0, scause
12 csrr a1, sepc
13 call trap_handler
14 # 3. restore sepc and 32 registers (x2(sp) should be restore last) from stack
15 # ld registers
16 ...
17 # addi sp, sp, 32*8
```

```
# 4. return from trap
sret
```

4、实现 trap 处理函数

1. 在 arch/riscv/kernel/ 目录下添加 trap.c 文件。
2. 在 trap.c 中实现 trap 处理函数 trap_handler(), 其接收的两个参数分别是 scause 和 sepc 两个寄存器中的值。

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2–4	*Reserved*
1	5	Supervisor timer interrupt
1	6–8	*Reserved*
1	9	Supervisor external interrupt
1	10–15	*Reserved*
1	≥16	*Designated for platform use*

如图，首位为1即中断，expcode为5即timer中断

```
#include "printk.h"
#include "clock.h"
void trap_handler(unsigned long scause, unsigned long sepc) {
    // 通过 `scause` 判断trap类型
    // 1. 如果是interrupt 判断是否是timer interrupt
    if ((scause & 0x8000000000000000) == 0x8000000000000000)//最高位为1，inter
    {
        // 1.1 如果是timer interrupt 则打印: "[S] Supervisor Mode Timer Interr
        //     并通过 `clock_set_next_event()` 设置下一次时钟中断
        if ((scause & 5) == 5)//最高位为1，exp code5
        {
            printk("[S] Supervisor Mode Timer Interrupt\n");
            clock_set_next_event();
        }
        // 1.2其他interrupt / exception 可以直接忽略
        else
            return;
    }
    //2.同1.2
```

```
    return;
}
```

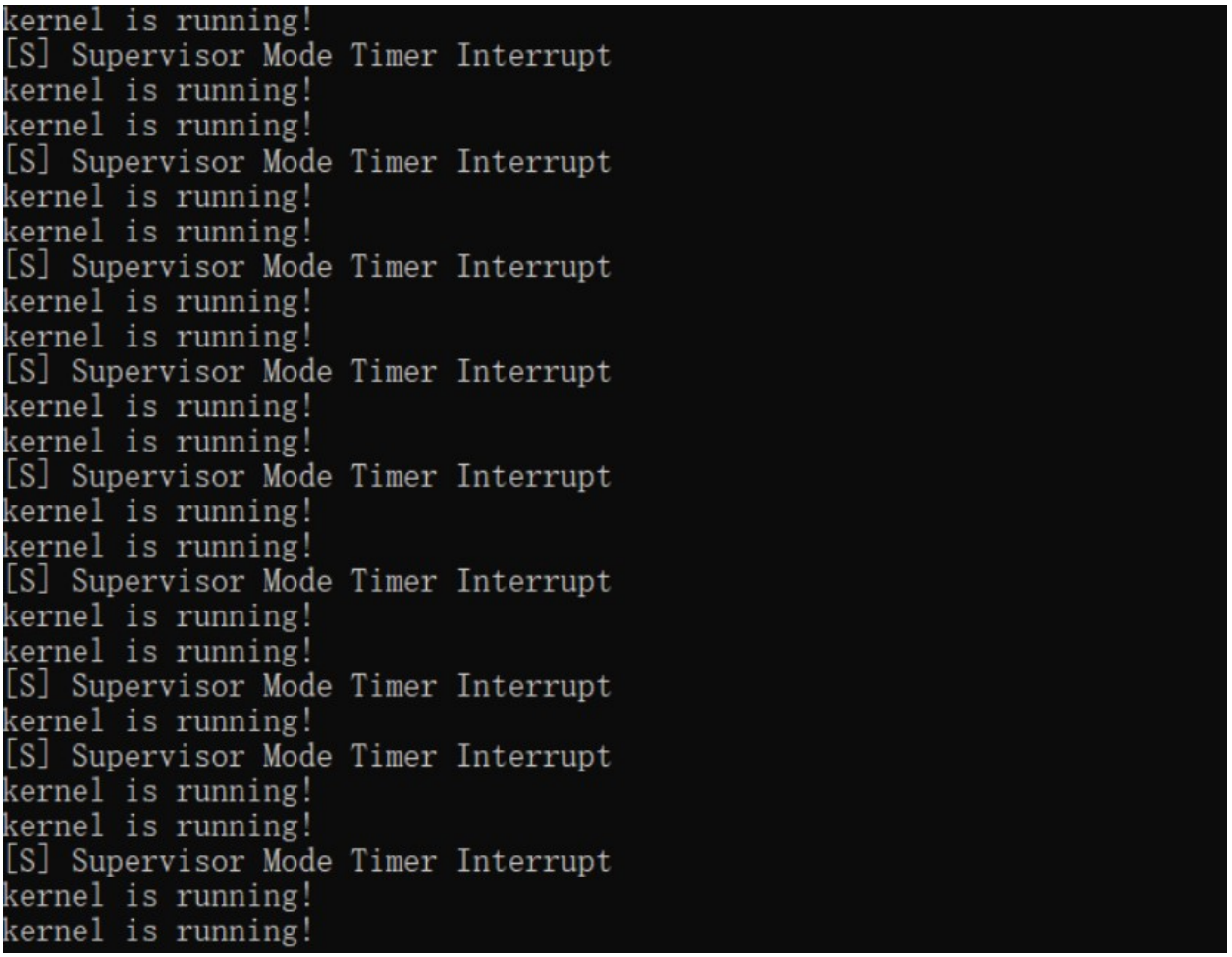
5、实现时钟中断相关函数

1. 在 arch/riscv/kernel/ 目录下添加 clock.c 文件。
2. 在 clock.c 中实现 get_cycles(): 使用 **rdtime** 汇编指令获得当前 time 寄存器中的值。
3. 在 clock.c 中实现 clock_set_next_event() : 调用 **sbi_ecall()**，设置下一个时钟中断事件。

```
#include "sbi.h"

// 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() {
    // 编写内联汇编，使用 rdtime 获取 time 寄存器中 (也就是mtime 寄存器 )的值并返
    // YOUR CODE HERE
    unsigned long time;
    asm volatile(
    "rdtime %[time]\n"
    : [time] "=r" (time)
    :
    :
);
    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
    // YOUR CODE HERE
    sbi_ecall(0, 0, next, 0, 0, 0, 0, 0);
}
```

6、编译及测试

make run输出：



gdb调试，_traps前后寄存器值：

```
(gdb) target remote :1234
Remote debugging using :1234
0x0000000000001000 in ?? ()
(gdb) set riscv use-compressed-breakpoints on
(gdb) b _traps
Breakpoint 1 at 0x8020004c: file entry.S, line 7.
(gdb) i r
ra             0x0      0x0
sp             0x0      0x0
gp             0x0      0x0
tp             0x0      0x0
t0             0x0      0
t1             0x0      0
t2             0x0      0
fp             0x0      0x0
s1             0x0      0
a0             0x0      0
a1             0x0      0
a2             0x0      0
a3             0x0      0
a4             0x0      0
a5             0x0      0
a6             0x0      0
a7             0x0      0
s2             0x0      0
s3             0x0      0
s4             0x0      0
s5             0x0      0
s6             0x0      0
s7             0x0      0
s8             0x0      0
s9             0x0      0
s10            0x0      0
s11            0x0      0
t3             0x0      0
t4             0x0      0
t5             0x0      0
t6             0x0      0
pc             0x1000   0x1000
(gdb)
```

continue之后查看寄存器：

```
(gdb) i r
ra             0x80200350       0x80200350 <start_kernel+48>
sp             0x80206fd0       0x80206fd0
gp             0x0      0x0
tp             0x80018000       0x80018000
t0             0x8000000000006002       -9223372036854751230
t1             0x0      0
t2             0x0      0
fp             0x80206ff0       0x80206ff0
s1             0x1      1
a0             0xe      14
a1             0x0      0
a2             0x0      0
a3             0xa      10
a4             0x39393b 3750203
a5             0x2626000        40001536
a6             0x0      0
a7             0x1      1
s2             0x8000000a00006800       -9223371993905076224
s3             0x80200000       2149580800
s4             0x87000000       2264924160
s5             0x0      0
s6             0x0      0
s7             0x800120e8       2147557608
s8             0x80013100       2147561728
s9             0x7f     127
s10            0x0      0
s11            0x0      0
t3             0x10     16
t4             0x80017ee0       2147581664
t5             0x27     39
t6             0x0      0
pc             0x8020004c       0x8020004c <_traps>
(gdb)
```

三、讨论和心得

本次实验主要是完成操作系统对trap和时钟中断的处理，实验过程中我们进一步理解了CPU上下文切换、trap和中断等机制，熟悉了riscv汇编指令以及gdb相关调试技巧。

我们在上下文切换部分，riscv汇编代码给C函数传参时遇到了困难。后来通过gdb调试发现将函数参数scause和sepc传入riscv的函数参数寄存器a0和a1（即x10和x11）即可。

四、思考题

在我们使用make run时， OpenSBI 会产生输出:

mideleg和medeleg两个寄存器与中断异常密切相关，主要用于将M模式的一些异常处理委托给S模式。

medeleg ： 异常委托寄存器。

mideleg：中断委托寄存器。

默认情况下，任何特权级别的所有陷阱都会在机器模式下处理，当然机器模式处理程序可以使用 MRET 指令将**陷阱**重定向回适当的模式级别。但是为了提高性能，RISC-V 提供了一种硬件机制，那就是**异常中断委托机制**。有了这个机制后，就不再需要软件程序上使用 MRET 指令将**陷阱**重定向回想要的模式级别。

medeleg 和 mideleg 寄存器中提供单独的读/写位，来指定某些异常和中断类型可以直接由某一较低的模式来处理。