lab1: 断, 都可以断

窦楷然 2210652 屈华晨 2210650

张梓恒 2211279

南开大学计算机学院

问题1：内核启动中的程序入口操作

文件：`kern/init/entry.S`

代码段

```
kern_entry:
    la sp, bootstacktop
    tail kern_init
```

解释

- `la sp, bootstacktop`：将堆栈指针 `sp` 初始化为 `bootstacktop` 的地址，确保内核可以使用一个合适的堆栈。该操作保证了后续函数调用和局部变量能够正确存储在内存中。
- `tail kern_init`：调用 `kern_init` 函数时使用 `tail` 优化，确保函数返回后不会执行多余的指令，节省处理时间，增强性能。

知识点

- **内核启动**：指的是操作系统从引导加载程序开始加载内核并初始化系统的过程，涉及堆栈设置、内存管理、设备初始化等。
- **OS原理中的对应知识点**：内核态与用户态的切换、系统初始化。
- **关系与差异**：实验中具体实现了内核的启动和堆栈初始化，理论知识则关注系统如何安全高效地完成启动过程。

练习2：完善中断处理

文件：`kern/trap/trap.c`

代码段

```
void interrupt_handler(struct trapframe *tf) {
    ...
    case IRQ_S_TIMER:
        clock_set_next_event();
        ticks++;
        if (ticks == 100) {
            print_ticks();
            ticks = 0;
            num++;
        }
        if (num == 10) {
            sbi_shutdown();
        }
        break;
}
```

解释

- `clock_set_next_event();`：设置下一个时钟中断事件，确保定时器能够准确触发下一个中断，保持时钟的准确性。
- `ticks++;`：每次时钟中断触发时，增加 `ticks` 计数，用于跟踪时钟中断的发生次数。
- `if (ticks == 100)`：当 `ticks` 达到100时，调用 `print_ticks()` 打印信息，提醒用户系统已运行一段时间，并重置 `ticks` 以准备下一个周期。
- `if (num == 10)`：当打印次数 `num` 达到10时，调用 `sbi_shutdown()` 进行关机，确保系统正常关闭。

知识点

- **中断处理**：操作系统响应硬件或软件中断的机制，确保系统能够处理多种事件，提升系统的响应性和并发能力。
- **OS原理中的对应知识点**：中断向量表、上下文切换、时钟中断的重要性。
- **关系与差异**：实验中实现了中断处理的具体实现，而理论知识涵盖中断的管理机制、上下文保存等。

扩展练习 Challenge1：描述与理解中断流程

文件：`kern/trap/trapentry.S`

代码段

```
1  mov a0, sp
2  SAVE_ALL
```

解释

- `mov a0, sp`：将当前堆栈指针 `sp` 复制到 `a0`，以便后续处理中访问堆栈信息，帮助调试和异常处理。
- `SAVE_ALL`：将当前所有寄存器的值保存在栈中，确保在中断处理完成后能够恢复寄存器状态，避免数据丢失。

知识点

- **中断流程**：中断发生时的处理流程，包括保存状态、执行中断服务例程以及恢复上下文。
- **OS原理中的对应知识点**：中断服务程序、上下文保存与恢复。
- **关系与差异**：实验中实现了中断的具体处理过程，而理论知识探讨中断的机制和优化策略。

扩展练习 Challenge2：理解上下文切换机制

文件： `kern/trap/trapentry.S`

代码段

```
1  csrw sscratch, sp
2  csrrw s0, sscratch, x0
```

解释

- `csrw sscratch, sp`：将当前堆栈指针 `sp` 保存到 CSR 寄存器 `sscratch` 中，方便在中断或异常发生后使用。
- `csrrw s0, sscratch, x0`：将 `sscratch` 中的值加载到 `s0` 中，同时清空 `sscratch`，确保寄存器状态一致，便于恢复。

知识点

- **上下文切换**：操作系统在不同进程或线程之间切换的机制，允许系统高效地运行多个任务。
- **OS原理中的对应知识点**：进程控制块、调度算法、调度策略的影响。
- **关系与差异**：实验中实现了上下文切换的指令操作，而理论知识侧重于调度策略及其性能评估。

扩展练习 Challenge3：完善异常中断

文件：`kern/trap/trap.c`

代码段

```
case CAUSE_ILLEGAL_INSTRUCTION:
    cprintf("Exception type: Illegal instruction\n");
    cprintf("Illegal instruction caught at 0x%08x\n", tf->epc);
    tf->epc += 4;
    break;
case CAUSE_BREAKPOINT:
    cprintf("Exception type: breakpoint\n");
    cprintf("ebreak caught at 0x%08x\n", tf->epc);
    tf->epc += 2;
    break;
```

解释

- **非法指令异常处理**：当检测到非法指令时，输出异常类型和触发地址，并更新 `tf->epc` 以跳过当前指令，使系统继续运行。
- **断点异常处理**：输出相关信息并更新 `tf->epc` ，通常加2是因为断点指令的长度，以确保在异常发生后继续执行后续指令。

知识点

- **异常处理**：操作系统对运行时错误的响应机制，确保系统能够处理各种异常情况。
- **OS原理中的对应知识点**：异常向量表、异常处理程序的设计。
- **关系与差异**：实验中实现了具体的异常处理逻辑，而理论知识探讨错误管理和恢复策略的设计。

额外的OS原理中重要但未在实验中对应的知识点

- **死锁管理**：实验主要集中在基本的中断和异常处理，但未涉及多进程环境中的死锁检测和解决机制。
- **虚拟内存管理**：实验关注内核启动和中断处理，而没有涉及虚拟内存的管理，包括页表、内存分配等。
- **文件系统管理**：未涵盖操作系统对文件系统的支持和管理，包括文件的读写、目录结构等。
