《操作系统原理与实践》
实验报告

| | | |
|---|---|---|
| 实验名称 | ： | Lab2：RV64 内核线程调度 |
| 姓　　名 | ： | 王晓宇 |
| 学　　号 | ： | 3220104364 |
| 电子邮箱 | ： | 3220104364@zju.edu.cn |
| 联系电话 | ： | 19550222634 |
| 授课教师 | ： | 申文博 |
| 助　　教 | ： | 陈淦豪&王鹤翔&许昊瑞 |

2024 年 10 月 9 日

**Lab2: RV64 内核线程调度**

# Lab2：RV64 内核线程调度

## 1 实验内容及简要原理介绍

- 了解线程概念，并学习线程相关结构体，并实现线程的初始化功能
- 了解如何使用时钟中断来实现线程的调度
- 了解线程切换原理，并实现线程的切换
- 掌握简单的线程调度算法，并完成简单调度算法的实现

## 2 实验具体过程与代码实现

### 2.1 准备工程4.1

已同步Lab1代码

### 2.1.1 在_start处调用mm_init:

src > lab2 > arch > riscv > kernel > head.S

```
    .extern start_kernel
    .extern _traps
    .extern sbi_set_timer
    .extern task_init
    .extern mm_init
    .section .text.init
    .globl _start
_start:
    la sp, boot_stack_top

    la t0, _traps
    csrw stvec, t0
    li t0, 0x20
    csrs sie, t0
    rdtime t0
    li t1, 10000000
    add a0, t0, t1
    call sbi_set_timer
    li t0, 0x2
    csrs sstatus, t0

    call mm_init
    call task_init
    call start_kernel

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

### 2.1.2 在 defs.h 中添加如下内容:

```
#define PHY_START 0x0000000080000000
#define PHY_SIZE 128 * 1024 * 1024 // 128 MiB, QEMU 默认内存大小
#define PHY_END (PHY_START + PHY_SIZE)

#define PGSIZE 0x1000 // 4 KiB
#define PGROUNDUP(addr) ((addr + PGSIZE - 1) & (~(PGSIZE - 1)))
#define PGROUNDDOWN(addr) (addr & (~(PGSIZE - 1)))
```

### 2.1.3 确保工程可以正常运行

```
Boot HART MIDELEG          : 0x0000000000001666
Boot HART MEDELEG          : 0x0000000000f0b509
...mm_init done!
...task_init done!
2024 ZJU Operating System
```

## 2.2 线程调度功能实现

### 2.2.1 线程初始化

```c
void task_init() {
    srand(2024);

    idle = (struct task_struct*)kalloc();
    idle->state = TASK_RUNNING;
    idle->counter = idle->priority = idle->pid = 0;
    current = task[0] = idle;

    for(int i = 1; i < NR_TASKS; i++) {
        task[i] = (struct task_struct*)kalloc();
        task[i]->state = TASK_RUNNING;
        task[i]->pid = i;
        task[i]->counter = 0;
        task[i]->priority = PRIORITY_MIN + rand() % (PRIORITY_MAX - PRIORITY_MIN + 1);
        task[i]->thread.ra = (uint64_t)&__dummy;
        task[i]->thread.sp = (uint64_t)task[i] + PGSIZE;
    }
    printk("...task_init done!\n");
}
```

- `srand(2024)` 初始化随机数生成器的种子，以确保每次运行程序时生成的随机数序列相同。
- 调用 [`kalloc`]分配一个物理页，并将其转换为 [`task_struct`] 类型的指针。
- 在后续的循环初始化线程中，使用 `rand` 函数生成一个随机数，并将其映射到 `[PRIORITY_MIN, PRIORITY_MAX]` 范围内，赋值给 `priority`。

### 2.2.2 __dummy 与 dummy 的实现

在 `arch/riscv/kernel/entry.S` 中添加函数 `__dummy`

```
    .extern dummy
    .globl __dummy
__dummy:
    la t0, dummy
    csrw sepc,t0
    sret
```

在 `__dummy` 中将 sepc 设置为 `dummy()` 的地址，并使用 `sret` 从 S 模式中返回

### 2.2.3 实现线程切换

```c
extern void __switch_to(struct task_struct *prev, struct task_struct *next);

void switch_to(struct task_struct *next) {
    if(current->pid != next->pid) {
        printk("switch to [PID = %d PRIORITY = %d COUNTER = %d]\n", next->pid, next->priority, next->counter);
        struct task_struct *prev = current;
        current = next;
        __switch_to(prev, next);
    }
}
```

- `prev`用于保存当前任务的指针，以便在切换任务时传递给 [`__switch_to`] 函数。
- `current`是一个全局变量，指向当前正在运行的任务。将其更新为 [`next`]，表示切换到下一个任务
- 调用汇编实现的`__switch_to` 函数，进行实际的任务上下文切换。

### 2.2.4 实现调度入口函数

```c
void do_timer() {
    if(current->pid == idle->pid || current->counter == 0) {
        schedule();
        return;
    } else {
        current->counter--;
    }
    if(current->counter <= 0) {
        schedule();
    }
}
```

1. 如果当前线程是 idle 线程或当前线程时间片耗尽则直接进行调度
2. 否则对当前线程的运行剩余时间减 1，若剩余时间仍然大于 0 则直接返回，否则进行调度

### 2.2.5 线程调度算法实现

```c
void schedule() {
    while(1){
        uint64_t max = 0;
        struct task_struct* max_task = NULL;
        for(int i = 0; i < NR_TASKS; i++) {
            if(task[i]->counter > max) {
                max = task[i]->counter;
                max_task = task[i];
            }
        }
        if(max == 0){
            for(int i = 1; i < NR_TASKS; i++) {
                task[i]->counter = task[i]->priority;
                printk("SET [PID = %d PRIORITY = %d COUNTER = %d]\n", task[i]->pid, task[i]->priority, task[i]->counter);
            }
            continue;
        } else {
```

```
            switch_to(max_task);
            return;
        }
    }
}
```

- `max`用于记录当前找到的最大优先级值，`max_task`用于记录具有最大计数器值的线程指针
- 通过遍历线程数组，比较每个线程的计数器值，如果找到更大的计数器值，则更新 `max`和 `max_task`。
- 如果 `max`仍然为 0，说明所有线程的计数器都已经用完，需要重新分配计数器。通过遍历线程数组，将每个线程的计数器值设置为其优先级，并打印调试信息。`continue`语句用于重新开始无限循环，重新选择线程。
- 如果找到计数器值最大的线程，则切换到该线程，并返回。

# 3 实验结果与分析

## 3.1 make TEST_SCHED=1 run

```
Boot HART MEDELEG          : 0x0000000000f0b509
...mm_init done!
...task_init done!
2024 ZJU Operating System
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 10 COUNTER = 10]
SET [PID = 3 PRIORITY = 4 COUNTER = 4]
SET [PID = 4 PRIORITY = 1 COUNTER = 1]
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
[PID = 2] is running. auto_inc_local_var = 1
[PID = 2] is running. auto_inc_local_var = 2
[PID = 2] is running. auto_inc_local_var = 3
[PID = 2] is running. auto_inc_local_var = 4
[PID = 2] is running. auto_inc_local_var = 5
[PID = 2] is running. auto_inc_local_var = 6
[PID = 2] is running. auto_inc_local_var = 7
[PID = 2] is running. auto_inc_local_var = 8
[PID = 2] is running. auto_inc_local_var = 9
[PID = 2] is running. auto_inc_local_var = 10
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[PID = 1] is running. auto_inc_local_var = 1
[PID = 1] is running. auto_inc_local_var = 2
[PID = 1] is running. auto_inc_local_var = 3
[PID = 1] is running. auto_inc_local_var = 4
[PID = 1] is running. auto_inc_local_var = 5
[PID = 1] is running. auto_inc_local_var = 6
[PID = 1] is running. auto_inc_local_var = 7
switch to [PID = 3 PRIORITY = 4 COUNTER = 4]
[PID = 3] is running. auto_inc_local_var = 1
[PID = 3] is running. auto_inc_local_var = 2
[PID = 3] is running. auto_inc_local_var = 3
[PID = 3] is running. auto_inc_local_var = 4
switch to [PID = 4 PRIORITY = 1 COUNTER = 1]
[PID = 4] is running. auto_inc_local_var = 1
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 10 COUNTER = 10]
SET [PID = 3 PRIORITY = 4 COUNTER = 4]
SET [PID = 4 PRIORITY = 1 COUNTER = 1]
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
[PID = 2] is running. auto_inc_local_var = 11
[PID = 2] is running. auto_inc_local_var = 12
[PID = 2] is running. auto_inc_local_var = 13
[PID = 2] is running. auto_inc_local_var = 14
[PID = 2] is running. auto_inc_local_var = 15
[PID = 2] is running. auto_inc_local_var = 16
[PID = 2] is running. auto_inc_local_var = 17
[PID = 2] is running. auto_inc_local_var = 18
[PID = 2] is running. auto_inc_local_var = 19
[PID = 2] is running. auto_inc_local_var = 20
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[PID = 1] is running. auto_inc_local_var = 8
[PID = 1] is running. auto_inc_local_var = 9
[PID = 1] is running. auto_inc_local_var = 10
[PID = 1] is running. auto_inc_local_var = 11
[PID = 1] is running. auto_inc_local_var = 12
[PID = 1] is running. auto_inc_local_var = 13
[PID = 1] is running. auto_inc_local_var = 14
switch to [PID = 3 PRIORITY = 4 COUNTER = 4]
[PID = 3] is running. auto_inc_local_var = 5
Test passed!
    Output: 2222222222111111133334222222222211111113
```

## 3.2 make run

```
...mm_init done!
...task_init done!
2024 ZJU Operating System
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 10 COUNTER = 10]
SET [PID = 3 PRIORITY = 4 COUNTER = 4]
SET [PID = 4 PRIORITY = 1 COUNTER = 1]
SET [PID = 5 PRIORITY = 4 COUNTER = 4]
SET [PID = 6 PRIORITY = 7 COUNTER = 7]
SET [PID = 7 PRIORITY = 5 COUNTER = 5]
SET [PID = 8 PRIORITY = 10 COUNTER = 10]
SET [PID = 9 PRIORITY = 1 COUNTER = 1]
SET [PID = 10 PRIORITY = 9 COUNTER = 9]
SET [PID = 11 PRIORITY = 6 COUNTER = 6]
SET [PID = 12 PRIORITY = 9 COUNTER = 9]
SET [PID = 13 PRIORITY = 6 COUNTER = 6]
SET [PID = 14 PRIORITY = 6 COUNTER = 6]
SET [PID = 15 PRIORITY = 5 COUNTER = 5]
SET [PID = 16 PRIORITY = 8 COUNTER = 8]
SET [PID = 17 PRIORITY = 1 COUNTER = 1]
SET [PID = 18 PRIORITY = 5 COUNTER = 5]
SET [PID = 19 PRIORITY = 3 COUNTER = 3]
SET [PID = 20 PRIORITY = 7 COUNTER = 7]
SET [PID = 21 PRIORITY = 7 COUNTER = 7]
SET [PID = 22 PRIORITY = 3 COUNTER = 3]
SET [PID = 23 PRIORITY = 3 COUNTER = 3]
SET [PID = 24 PRIORITY = 3 COUNTER = 3]
SET [PID = 25 PRIORITY = 4 COUNTER = 4]
SET [PID = 26 PRIORITY = 3 COUNTER = 3]
SET [PID = 27 PRIORITY = 9 COUNTER = 9]
SET [PID = 28 PRIORITY = 1 COUNTER = 1]
SET [PID = 29 PRIORITY = 9 COUNTER = 9]
SET [PID = 30 PRIORITY = 10 COUNTER = 10]
SET [PID = 31 PRIORITY = 3 COUNTER = 3]
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
[PID = 2] is running. auto_inc_local_var = 1
[PID = 2] is running. auto_inc_local_var = 2
[PID = 2] is running. auto_inc_local_var = 3
[PID = 2] is running. auto_inc_local_var = 4
[PID = 2] is running. auto_inc_local_var = 5
[PID = 2] is running. auto_inc_local_var = 6
[PID = 2] is running. auto_inc_local_var = 7
[PID = 2] is running. auto_inc_local_var = 8
[PID = 2] is running. auto_inc_local_var = 9
[PID = 2] is running. auto_inc_local_var = 10
switch to [PID = 8 PRIORITY = 10 COUNTER = 10]
[PID = 8] is running. auto_inc_local_var = 1
[PID = 8] is running. auto_inc_local_var = 2
[PID = 8] is running. auto_inc_local_var = 3
[PID = 8] is running. auto_inc_local_var = 4
[PID = 8] is running. auto_inc_local_var = 5
[PID = 8] is running. auto_inc_local_var = 6
[PID = 8] is running. auto_inc_local_var = 7
[PID = 8] is running. auto_inc_local_var = 8
[PID = 8] is running. auto_inc_local_var = 9
[PID = 8] is running. auto_inc_local_var = 10
switch to [PID = 30 PRIORITY = 10 COUNTER = 10]
```

```
[PID = 30] is running. auto_inc_local_var = 1
[PID = 30] is running. auto_inc_local_var = 2
[PID = 30] is running. auto_inc_local_var = 3
[PID = 30] is running. auto_inc_local_var = 4
[PID = 30] is running. auto_inc_local_var = 5
[PID = 30] is running. auto_inc_local_var = 6
[PID = 30] is running. auto_inc_local_var = 7
[PID = 30] is running. auto_inc_local_var = 8
[PID = 30] is running. auto_inc_local_var = 9
[PID = 30] is running. auto_inc_local_var = 10
switch to [PID = 10 PRIORITY = 9 COUNTER = 9]
[PID = 10] is running. auto_inc_local_var = 1
[PID = 10] is running. auto_inc_local_var = 2
[PID = 10] is running. auto_inc_local_var = 3
[PID = 10] is running. auto_inc_local_var = 4
[PID = 10] is running. auto_inc_local_var = 5
[PID = 10] is running. auto_inc_local_var = 6
[PID = 10] is running. auto_inc_local_var = 7
[PID = 10] is running. auto_inc_local_var = 8
[PID = 10] is running. auto_inc_local_var = 9
switch to [PID = 12 PRIORITY = 9 COUNTER = 9]
[PID = 12] is running. auto_inc_local_var = 1
[PID = 12] is running. auto_inc_local_var = 2
[PID = 12] is running. auto_inc_local_var = 3
[PID = 12] is running. auto_inc_local_var = 4
[PID = 12] is running. auto_inc_local_var = 5
[PID = 12] is running. auto_inc_local_var = 6
[PID = 12] is running. auto_inc_local_var = 7
[PID = 12] is running. auto_inc_local_var = 8
[PID = 12] is running. auto_inc_local_var = 9
switch to [PID = 27 PRIORITY = 9 COUNTER = 9]
[PID = 27] is running. auto_inc_local_var = 1
[PID = 27] is running. auto_inc_local_var = 2
[PID = 27] is running. auto_inc_local_var = 3
[PID = 27] is running. auto_inc_local_var = 4
[PID = 27] is running. auto_inc_local_var = 5
[PID = 27] is running. auto_inc_local_var = 6
[PID = 27] is running. auto_inc_local_var = 7
[PID = 27] is running. auto_inc_local_var = 8
[PID = 27] is running. auto_inc_local_var = 9
switch to [PID = 29 PRIORITY = 9 COUNTER = 9]
[PID = 29] is running. auto_inc_local_var = 1
[PID = 29] is running. auto_inc_local_var = 2
[PID = 29] is running. auto_inc_local_var = 3
[PID = 29] is running. auto_inc_local_var = 4
[PID = 29] is running. auto_inc_local_var = 5
[PID = 29] is running. auto_inc_local_var = 6
[PID = 29] is running. auto_inc_local_var = 7
[PID = 29] is running. auto_inc_local_var = 8
[PID = 29] is running. auto_inc_local_var = 9
switch to [PID = 16 PRIORITY = 8 COUNTER = 8]
[PID = 16] is running. auto_inc_local_var = 1
[PID = 16] is running. auto_inc_local_var = 2
[PID = 16] is running. auto_inc_local_var = 3
[PID = 16] is running. auto_inc_local_var = 4
[PID = 16] is running. auto_inc_local_var = 5
[PID = 16] is running. auto_inc_local_var = 6
[PID = 16] is running. auto_inc_local_var = 7
[PID = 16] is running. auto_inc_local_var = 8
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[PID = 1] is running. auto_inc_local_var = 1
```

# 4 思考题与心得体会

1. 在 RV64 中一共有 32 个通用寄存器，为什么 `__switch_to` 中只保存了 14 个？

   根据 RISC-V 的调用约定，有些寄存器是调用者保存的（caller-saved），有些是被调用者保存的（callee-saved）。在上下文切换时，只需要保存被调用者保存的寄存器，通过只保存必要的寄存器，可以提高上下文切换的效率。

   | 寄存器 | ABI 名称 | 用途描述 | saver |
   |---|---|---|---|
   | x0 | zero | 硬件 0 | |
   | x1 | ra | 返回地址（return address） | caller |
   | x2 | sp | 栈指针（stack pointer） | callee |
   | x3 | gp | 全局指针（global pointer） | |
   | x4 | tp | 线程指针（thread pointer） | |
   | x5 | t0 | 临时变量 / 备用链接寄存器（alternate link reg） | caller |
   | x6-7 | t1-2 | 临时变量 | caller |
   | x8 | s0/fp | 需要保存的寄存器 / 帧指针（frame pointer） | callee |
   | x9 | s1 | 需要保存的寄存器 | callee |
   | x10-11 | a0-1 | 函数参数 / 返回值 | caller |
   | x12-17 | a2-7 | 函数参数 | caller |
   | x18-27 | s2-11 | 需要保存的寄存器 | callee |
   | x28-31 | t3-6 | 临时变量 | caller |

   其中 sp s0-11 需要在函数调用前后保证一致，其它不用保证

   > 图片来源[RISC-V 非特权级 ISA - 鹤翔万里的笔记本 (tonycrane.cc)]

   `sp` `s0-11` 是`callee saved`需要在函数调用前后保证一致，而我们这里又保存了ra方便进程重新恢复执行的PC.

2. 阅读并理解 `arch/riscv/kernel/mm.c` 代码，尝试说明 `mm_init` 函数都做了什么，以及在 `kalloc` 和 `kfree` 的时候内存是如何被管理的。

在 `mm_init` 函数中，通过调用 `kfreerange` 函数(将起始地址对齐到页大小`PGSIZE`,从起始地址开始，逐页调用 `kfree` 函数，直到结束地址)，将从内核结束地址 `_ekernel` 到物理内存结束地址 `PHY_END` 之间的所有内存页释放到空闲内存页链表中以便调出使用。

> 在 `kalloc` 和 `kfree` 的时候内存主要是通过分配好的`kmem.freelist`来进行管理

**`kalloc` 函数用于分配一个内存页。**

1. 从空闲内存页链表 `kmem.freelist` 中取出一个内存页。
2. 将该内存页的内容清零，并返回该内存页的地址以实现分配一个内存页的效果。

**`kfree` 函数用于释放一个内存页。**

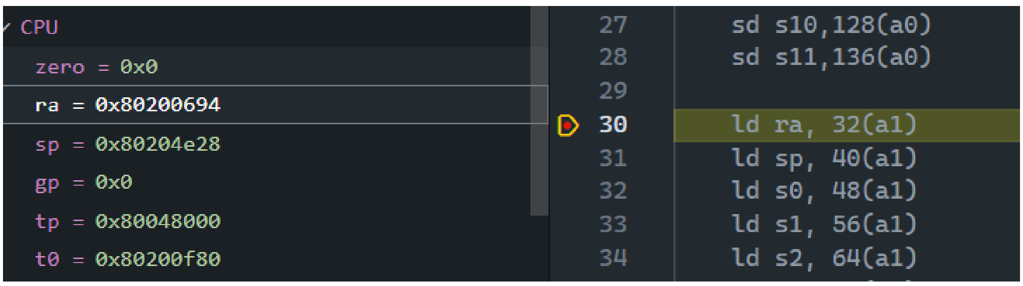
1. 将传入的地址 `addr` 对齐到页大小（`PGSIZE`）。将该内存页的内容清零。
2. 将该内存页添加到空闲内存页链表 `kmem.freelist` 的头部，相当于回收一页内存并标记为释放，链接到`list`的头部以便以后分配使用。

3. 当线程第一次调用时，其 `ra` 所代表的返回点是 `__dummy`，那么在之后的线程调用中 `__switch_to` 中，`ra` 保存 / 恢复的函数返回点是什么呢？请同学用 gdb 尝试追踪一次完整的线程切换流程，并关注每一次 `ra` 的变换（需要截图）。

`ra` 保存 / 恢复的函数返回点是`__switch`,我们可以在`make TEST_SCHED=1 debug`模式中观察PID = 2的线程。

**当PID=2的线程第一次被调用时：**

[PID = 2 PRIORITY = 10 COUNTER = 10]



```
Registers
  CPU
    zero = 0x0
    ra = 0x80200044
    sp = 0x80204e28
    gp = 0x0
    tp = 0x80048000
    t0 = 0x80200f78
    t1 = 0x0
    t2 = 0x0
    fp = 0x80204e58
```

```
    sd s9,120(a0)
    sd s10,128(a0)
    sd s11,136(a0)

    ld ra, 32(a1)
    ld sp, 40(a1)
    ld s0, 48(a1)
    ld s1, 56(a1)
    ld s2, 64(a1)
    ld s3, 72(a1)
    ld s4, 80(a1)
    ld s5, 88(a1)
```

ra = 0x80200694 → 0x80200044

```
void switch_to(struct task_struct *next) {
    8020060c:	fd010113          	addi	sp,sp,-48
    80200610:	02113423          	sd	ra,40(sp)
    80200614:	02813023          	sd	s0,32(sp)
    80200618:	03010413          	addi	s0,sp,48
    8020061c:	fca43c23          	sd	a0,-40(s0)
    if(current->pid != next->pid) {
    80200620:	00005797          	auipc	a5,0x5
    80200624:	9f078793          	addi	a5,a5,-1552 # 80205010 <current>
    80200628:	0007b783          	ld	a5,0(a5)
    8020062c:	0187b703          	ld	a4,24(a5)
    80200630:	fd843783          	ld	a5,-40(s0)
    80200634:	0187b783          	ld	a5,24(a5)
    80200638:	04f70e63          	beq	a4,a5,80200694 <switch_to+0x88>
        printk("switch to [PID = %d PRIORITY = %d COUNTER = %d]\n", next->pid, n
    8020063c:	fd843783          	ld	a5,-40(s0)
    80200640:	0187b703          	ld	a4,24(a5)
    80200644:	fd843783          	ld	a5,-40(s0)
    80200648:	0107b603          	ld	a2,16(a5)
    8020064c:	fd843783          	ld	a5,-40(s0)
    80200650:	0087b783          	ld	a5,8(a5)
    80200654:	00078693          	mv	a3,a5
    80200658:	00070593          	mv	a1,a4
    8020065c:	00002517          	auipc	a0,0x2
    80200660:	9d450513          	addi	a0,a0,-1580 # 80202030 <_srodata+0x3
    80200664:	7c8010ef          	jal	ra,80201e2c <printk>
        struct task_struct *prev = current;
    80200668:	00005797          	auipc	a5,0x5
    8020066c:	9a878793          	addi	a5,a5,-1624 # 80205010 <current>
    80200670:	0007b783          	ld	a5,0(a5)
    80200674:	fef43423          	sd	a5,-24(s0)
        current = next;
    80200678:	00005797          	auipc	a5,0x5
    8020067c:	99878793          	addi	a5,a5,-1640 # 80205010 <current>
    80200680:	fd843703          	ld	a4,-40(s0)
    80200684:	00e7b023          	sd	a4,0(a5)
        __switch_to(prev, next);
    80200688:	fd843583          	ld	a1,-40(s0)
    8020068c:	fe843503          	ld	a0,-24(s0)
    80200690:	9c5ff0ef          	jal	ra,80200054 <__switch_to>
    }
}
    80200694:	00000013          	nop
    80200698:	02813083          	ld	ra,40(sp)
    8020069c:	02013403          	ld	s0,32(sp)
    802006a0:	03010113          	addi	sp,sp,48
    802006a4:	00008067          	ret
```

0x80200694是调用`__switch_to`后的指令地址

```
0000000080200044 <__dummy>:
    .extern dummy
    .globl __dummy
    .globl __switch_to

__dummy:
    la t0, dummy
    80200044:   00003297            auipc   t0,0x3
    80200048:   0142b283            ld  t0,20(t0) # 80203058 <_GLOBAL_OFFSET_TABL
    csrw sepc,t0
    8020004c:   14129073            csrw    sepc,t0
    sret
    80200050:   10200073            sret
```

在`vmlinux.lds`中可以看到0x80200044为`__dummy`的地址，说明线程被第一次调用时ra被赋值为我们指定的函数`__dummy`

**当PID=2的线程第二次被调用时：**

```
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
[PID = 2] is running. auto_inc_local_var = 1
[PID = 2] is running. auto_inc_local_var = 2
[PID = 2] is running. auto_inc_local_var = 3
[PID = 2] is running. auto_inc_local_var = 4
[PID = 2] is running. auto_inc_local_var = 5
[PID = 2] is running. auto_inc_local_var = 6
[PID = 2] is running. auto_inc_local_var = 7
[PID = 2] is running. auto_inc_local_var = 8
[PID = 2] is running. auto_inc_local_var = 9
[PID = 2] is running. auto_inc_local_var = 10
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
[PID = 1] is running. auto_inc_local_var = 1
[PID = 1] is running. auto_inc_local_var = 2
[PID = 1] is running. auto_inc_local_var = 3
[PID = 1] is running. auto_inc_local_var = 4
[PID = 1] is running. auto_inc_local_var = 5
[PID = 1] is running. auto_inc_local_var = 6
[PID = 1] is running. auto_inc_local_var = 7
switch to [PID = 3 PRIORITY = 4 COUNTER = 4]
[PID = 3] is running. auto_inc_local_var = 1
[PID = 3] is running. auto_inc_local_var = 2
[PID = 3] is running. auto_inc_local_var = 3
[PID = 3] is running. auto_inc_local_var = 4
switch to [PID = 4 PRIORITY = 1 COUNTER = 1]
[PID = 4] is running. auto_inc_local_var = 1
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 10 COUNTER = 10]
SET [PID = 3 PRIORITY = 4 COUNTER = 4]
SET [PID = 4 PRIORITY = 1 COUNTER = 1]
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
```

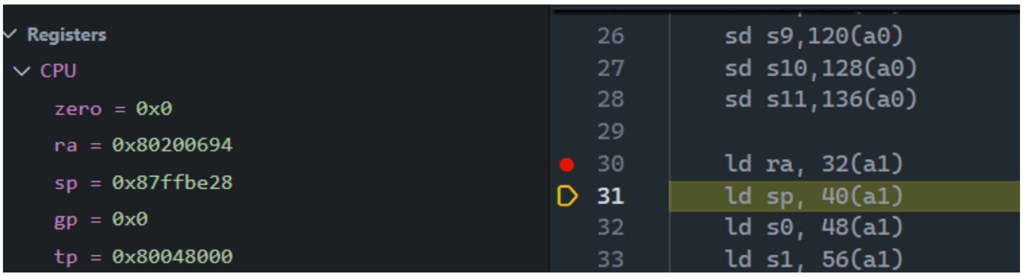
现在是第二次调用

`switch to [PID = 2 PRIORITY = 10 COUNTER = 10]`

```
CPU
  zero = 0x0
  ra = 0x80200694
  sp = 0x87ffbe28
  gp = 0x0
  tp = 0x80048000
```

```
sd s10,128(a0)
sd s11,136(a0)

ld ra, 32(a1)
ld sp, 40(a1)
ld s0, 48(a1)
ld s1, 56(a1)
```

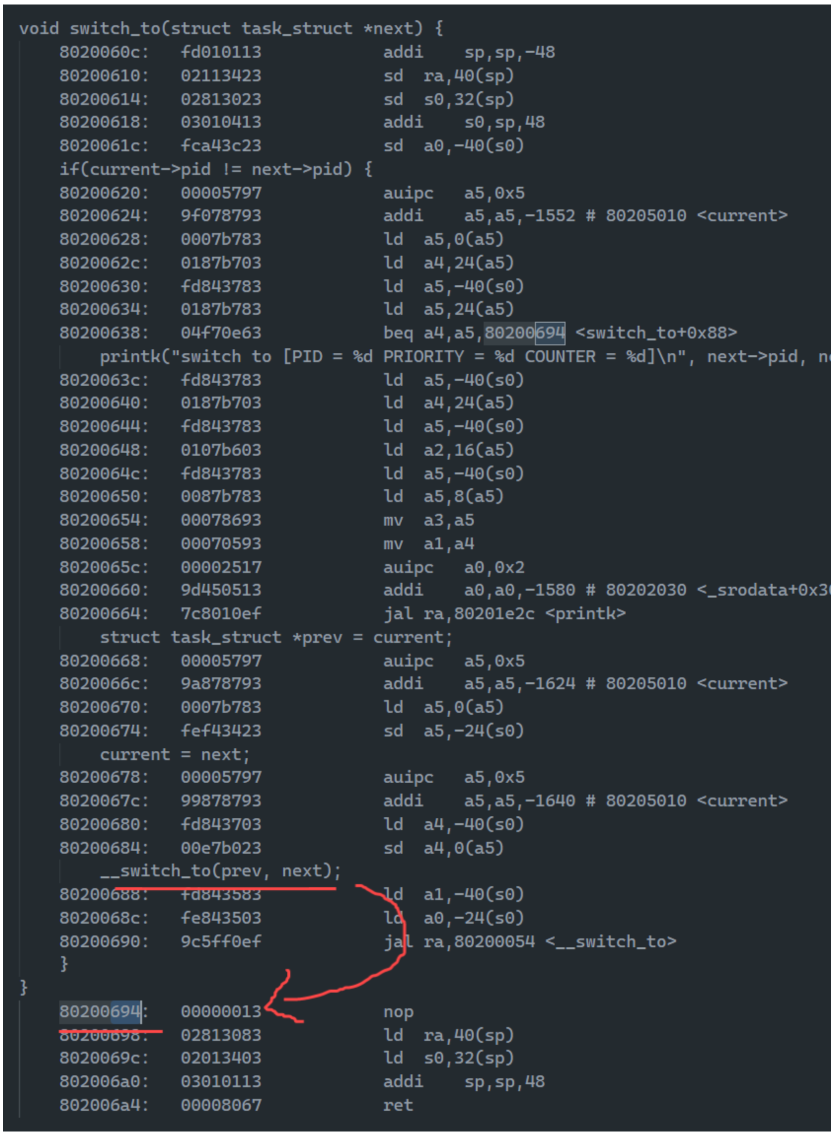


```
Registers
CPU
zero = 0x0
ra = 0x80200694
sp = 0x87ffbe28
gp = 0x0
tp = 0x80048000
```

```
sd s9,120(a0)
sd s10,128(a0)
sd s11,136(a0)

ld ra, 32(a1)
ld sp, 40(a1)
ld s0, 48(a1)
ld s1, 56(a1)
```

ra = 0x80200694 → 0x80200694



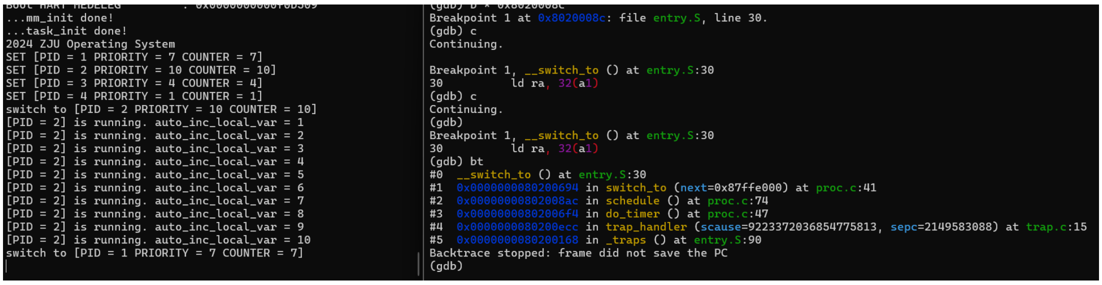
```
void switch_to(struct task_struct *next) {
    8020060c:	fd010113          	addi	sp,sp,-48
    80200610:	02113423          	sd	ra,40(sp)
    80200614:	02813023          	sd	s0,32(sp)
    80200618:	03010413          	addi	s0,sp,48
    8020061c:	fca43c23          	sd	a0,-40(s0)
    if(current->pid != next->pid) {
    80200620:	00005797          	auipc	a5,0x5
    80200624:	9f078793          	addi	a5,a5,-1552 # 80205010 <current>
    80200628:	0007b783          	ld	a5,0(a5)
    8020062c:	0187b703          	ld	a4,24(a5)
    80200630:	fd843783          	ld	a5,-40(s0)
    80200634:	0187b783          	ld	a5,24(a5)
    80200638:	04f70e63          	beq	a4,a5,80200694 <switch_to+0x88>
        printk("switch to [PID = %d PRIORITY = %d COUNTER = %d]\n", next->pid, n
    8020063c:	fd843783          	ld	a5,-40(s0)
    80200640:	0187b703          	ld	a4,24(a5)
    80200644:	fd843783          	ld	a5,-40(s0)
    80200648:	0107b603          	ld	a2,16(a5)
    8020064c:	fd843783          	ld	a5,-40(s0)
    80200650:	0087b783          	ld	a5,8(a5)
    80200654:	00078693          	mv	a3,a5
    80200658:	00070593          	mv	a1,a4
    8020065c:	00002517          	auipc	a0,0x2
    80200660:	9d450513          	addi	a0,a0,-1580 # 80202030 <_srodata+0x3
    80200664:	7c8010ef          	jal	ra,80201e2c <printk>
        struct task_struct *prev = current;
    80200668:	00005797          	auipc	a5,0x5
    8020066c:	9a878793          	addi	a5,a5,-1624 # 80205010 <current>
    80200670:	0007b783          	ld	a5,0(a5)
    80200674:	fef43423          	sd	a5,-24(s0)
        current = next;
    80200678:	00005797          	auipc	a5,0x5
    8020067c:	99878793          	addi	a5,a5,-1640 # 80205010 <current>
    80200680:	fd843703          	ld	a4,-40(s0)
    80200684:	00e7b023          	sd	a4,0(a5)
        __switch_to(prev, next);
    80200688:	fd843583          	ld	a1,-40(s0)
    8020068c:	fe843503          	ld	a0,-24(s0)
    80200690:	9c5ff0ef          	jal	ra,80200054 <__switch_to>
    }
}
    80200694:	00000013          	nop
    80200698:	02813083          	ld	ra,40(sp)
    8020069c:	02013403          	ld	s0,32(sp)
    802006a0:	03010113          	addi	sp,sp,48
    802006a4:	00008067          	ret
```

0x80200694是调用`__switch_to`后的指令地址，是上一次保存的上下文，因为线程从上一次调用`__switch_to`后就暂停执行了，现在切换回来开始执行之后的指令。

4. 请尝试分析并画图说明kernel运行到输出第两次`switch to [PID ...`的时候内存中存在的全部函数帧栈布局。

- 可通过 gdb 调试使用 `backtrace` 等指令辅助分析，注意分析第一次时钟中断触发后的 `pc` 和 `sp` 的变化。

这里我选择断地址断在 `__switch_to` 的切换上下文位置，利用 `backtrace` 查看调用帧栈：


```
...mm_init done!
...task_init done!
2024 ZJU Operating System
SET [PID = 1 PRIORITY = 7 COUNTER = 7]
SET [PID = 2 PRIORITY = 10 COUNTER = 10]
SET [PID = 3 PRIORITY = 4 COUNTER = 4]
SET [PID = 4 PRIORITY = 1 COUNTER = 1]
switch to [PID = 2 PRIORITY = 10 COUNTER = 10]
[PID = 2] is running. auto_inc_local_var = 1
[PID = 2] is running. auto_inc_local_var = 2
[PID = 2] is running. auto_inc_local_var = 3
[PID = 2] is running. auto_inc_local_var = 4
[PID = 2] is running. auto_inc_local_var = 5
[PID = 2] is running. auto_inc_local_var = 6
[PID = 2] is running. auto_inc_local_var = 7
[PID = 2] is running. auto_inc_local_var = 8
[PID = 2] is running. auto_inc_local_var = 9
[PID = 2] is running. auto_inc_local_var = 10
switch to [PID = 1 PRIORITY = 7 COUNTER = 7]
```

```
Breakpoint 1 at 0x8020008c: file entry.S, line 30.
(gdb) c
Continuing.

Breakpoint 1, __switch_to () at entry.S:30
30          ld ra, 32(a1)
(gdb) c
Continuing.
(gdb)
Breakpoint 1, __switch_to () at entry.S:30
30          ld ra, 32(a1)
(gdb) bt
#0  __switch_to () at entry.S:30
#1  0x0000000080200694 in switch_to (next=0x87ffe000) at proc.c:41
#2  0x00000000802008ac in schedule () at proc.c:74
#3  0x00000000802006f4 in do_timer () at proc.c:47
#4  0x0000000080200ecc in trap_handler (scause=9223372036854775813, sepc=2149583088) at trap.c:15
#5  0x0000000080200168 in _traps () at entry.S:90
Backtrace stopped: frame did not save the PC
(gdb)
```