Lab 2: RV64 内核线程调度

1 实验目的

- 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, 并完成两种简单调度算法的实现。

2 实验环境

- Environment in previous labs

3 背景知识

3.0 前言

在 lab1 中, 我们利用 trap 赋予了 OS 与软件, 硬件的交互能力。但是目前我们的 OS 还不具备多进程调度以及并发执行的能力。在本次实验中, 我们将利用时钟中断, 来实现多进程的调度以使得多个进程/线程并发执行。

3.1 进程与线程

`源代码` 经编译器一系列处理（编译、链接、优化等）后得到的可执行文件, 我们称之为 `程序(Program)`。而通俗地说, `进程` 就是 `正在运行并使用计算机资源` 的程序。 `进程` 与 `程序` 的不同之处在于, `进程` 是一个动态的概念, 其不仅需要将其运行的程序的代码/数据等加载到内存空间中, 还需要拥有自己的 `运行栈`。同时一个 `进程` 可以对应一个或多个 `线程`, `线程` 之间往往具有相同的代码, 共享一块内存, 但是却有不同的CPU执行状态。

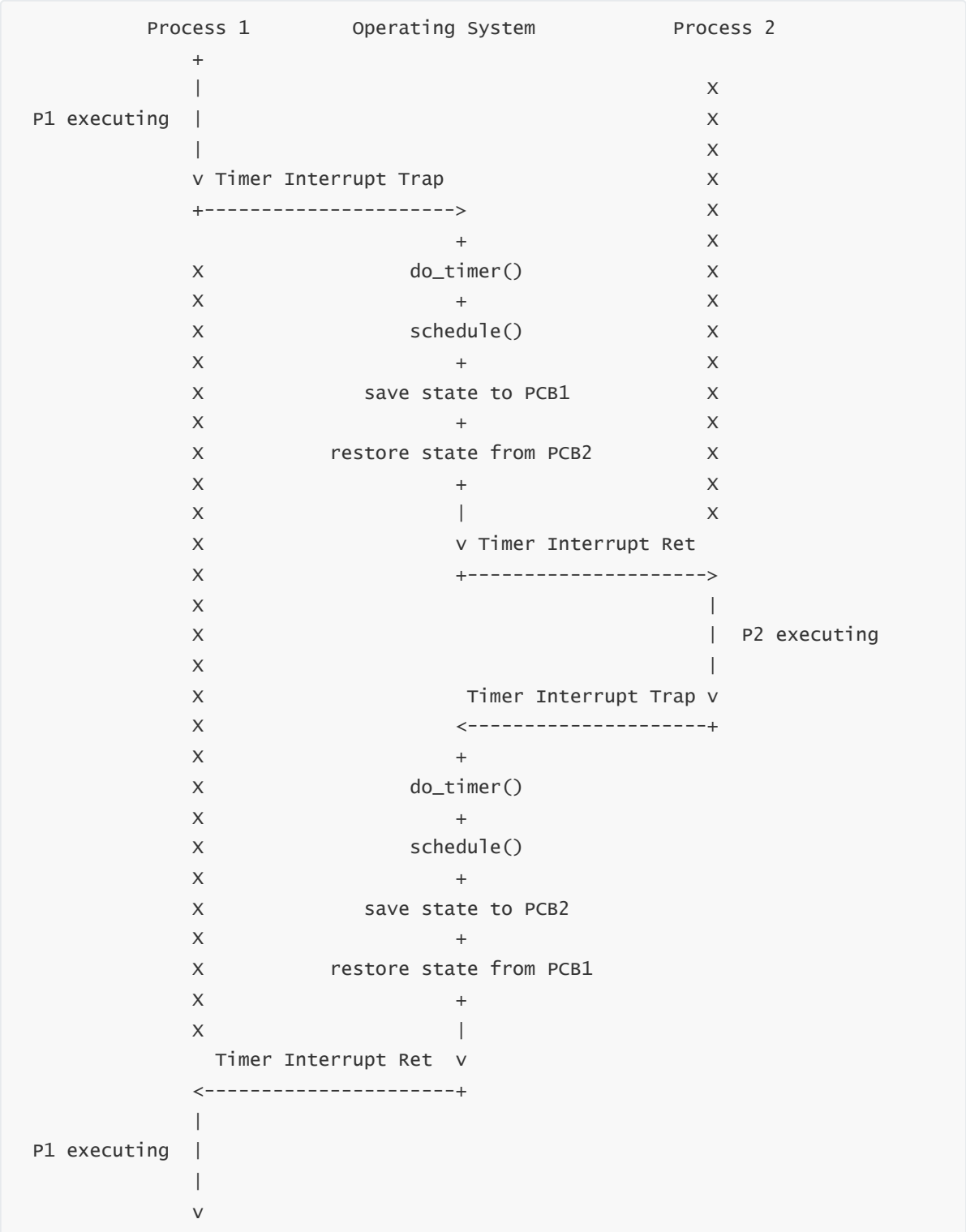
在本次实验中, 为了简单起见, 我们采用 `single-threaded process` 模型, 即 `一个进程` 对应 `一个线程`, 进程与线程不做明显区分。

3.1 线程相关属性

在不同的操作系统中, 为每个线程所保存的信息都不同。在这里, 我们提供一种基础的实现, 每个线程会包括:

- `线程ID`: 用于唯一确认一个线程。
- `运行栈`: 每个线程都必须有一个独立的运行栈, 保存运行时的数据。
- `执行上下文`: 当线程不在执行状态时, 我们需要保存其上下文（其实就是 `状态寄存器` 的值）, 这样之后才能够将其恢复, 继续运行。
- `运行时间片`: 为每个线程分配的运行时间。
- `优先级`: 在优先级相关调度时, 配合调度算法, 来选出下一个执行的线程。

3.2 线程切换流程图



- 在每次处理时钟中断时, 操作系统首先会将当前线程的运行剩余时间减少一个单位。之后根据调度算法来确定是继续运行还是调度其他线程来执行。
- 在进程调度时, 操作系统会遍历所有可运行的线程, 按照一定的调度算法选出下一个执行的线程。最终将选择得到的线程与当前线程切换。
- 在切换的过程中, 首先我们需要保存当前线程的执行上下文, 再将将要执行线程的上下文载入到相关寄存器中, 至此我们就完成了线程的调度与切换。

4 实验步骤

4.1 准备工程

- 此次实验基于 lab2 同学所实现的代码进行。
- 从 `repo` 同步以下代码: `rand.h/rand.c`, `string.h/string.c`, `mm.h/mm.c`。并按照以下步骤将这些文件正确放置。其中 `mm.h/mm.c` 提供了简单的物理内存管理接口, `rand.h/rand.c` 提供了 `rand()` 接口用以提供伪随机数序列, `string.h/string.c` 提供了 `memset` 接口用以初始化一段内存空间。

```
.
├── arch
│   └── riscv
│       ├── include
│       │   └── mm.h
│       └── kernel
│           └── mm.c
├── include
│   ├── rand.h
│   └── string.h
└── lib
    ├── rand.c
    └── string.c
```

- 在 lab3 中我们需要一些物理内存管理的接口, 在此我们提供了 `kalloc` 接口 (见 `mm.c`) 给同学。同学可以用 `kalloc` 来申请 4KB 的物理页。由于引入了简单的物理内存管理, 需要在 `_start` 的适当位置调用 `mm_init`, 来初始化内存管理系统, 并且在初始化时需要用一些自定义的宏, 需要修改 `defs.h`, 在 `defs.h` 添加如下内容:

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

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

- 请在添加/修改上述文件代码之后, 确保工程可以正常运行, 之后再开始实现 `lab3` (有可能需要同学自己调整一些头文件的引入)。
- 在 lab3 中需要同学需要添加并修改 `arch/riscv/include/proc.h` `arch/riscv/kernel/proc.c` 两个文件。
- 本次实验需要实现两种不同的调度算法, 如何控制代码逻辑见 `4.4`

4.2 `proc.h` 数据结构定义

```
// arch/riscv/include/proc.h

#include "types.h"

#define NR_TASKS  (1 + 31) // 用于控制 最大线程数量 （idle 线程 + 31 内核线程）
```

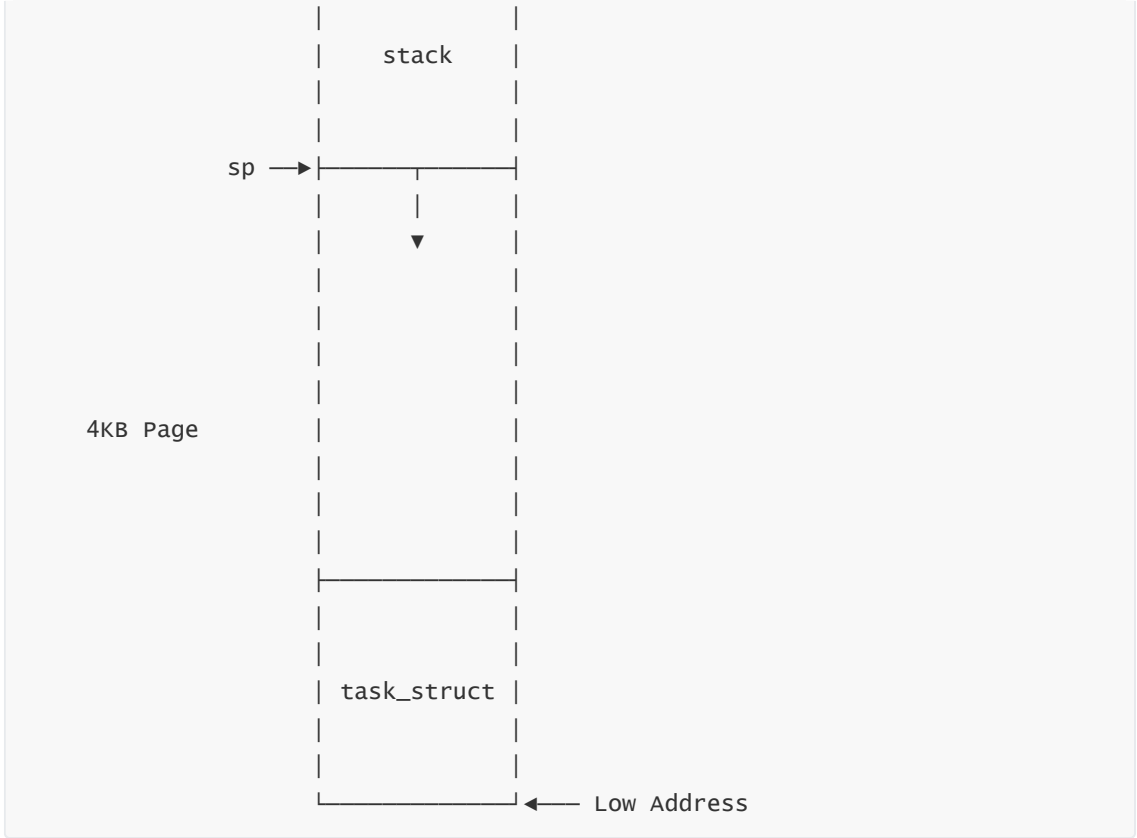
```
#define TASK_RUNNING    0 // 为了简化实验, 所有的线程都只有一种状态

#define PRIORITY_MIN 1
#define PRIORITY_MAX 10

/* 用于记录 `线程` 的 `内核栈与用户栈指针` */
/* (lab3中无需考虑, 在这里引入是为了之后实验的使用) */
struct thread_info {
    uint64 kernel_sp;
    uint64 user_sp;
};

/* 线程状态段数据结构 */
struct thread_struct {
    uint64 ra;
    uint64 sp;
    uint64 s[12];
};

/* 线程数据结构 */
struct task_struct {
    struct thread_info* thread_info;
    uint64 state;    // 线程状态
    uint64 counter;  // 运行剩余时间
    uint64 priority; // 运行优先级 1最低 10最高
    uint64 pid;      // 线程id

    struct thread_struct thread;
};

/* 线程初始化 创建 NR_TASKS 个线程 */
void task_init();

/* 在时钟中断处理中被调用 用于判断是否需要进行调度 */
void do_timer();

/* 调度程序 选择出下一个运行的线程 */
void schedule();

/* 线程切换入口函数*/
void switch_to(struct task_struct* next);

/* dummy funciton: 一个循环程序, 循环输出自己的 pid 以及一个自增的局部变量 */
void dummy();
```

4.3 线程调度功能实现

4.3.1 线程初始化

- 在初始化线程的时候, 我们参考Linux v0.11中的实现为每个线程分配一个 4KB 的物理页, 我们将 `task_struct` 存放在该页的低地址部分, 将线程的栈指针 `sp` 指向该页的高地址。具体内存布局如下图所示:

```
                                ┌──────────────┐◄─── High Address
```



- 当我们的 OS run 起来的时候, 其本身就是一个线程 `idle 线程`, 但是我们并没有为它设计好 `task_struct`。所以第一步我们要为 `idle` 设置 `task_struct`。并将 `current`, `task[0]` 都指向 `idle`。
- 为了方便起见, 我们将 `task[1]` ~ `task[NR_TASKS - 1]`, 全部初始化, 这里和 `idle` 设置的区别在于要为这些线程设置 `thread_struct` 中的 `ra` 和 `sp`.
- 在 `_start` 适当的位置调用 `task_init`

```
void task_init() {

    uint64 addr_idle = kalloc(); // 调用 kalloc() 为 idle 分配一个物理页
    idle = (struct task_struct*)addr_idle;
    idle->state = TASK_RUNNING; // 设置 state 为 TASK_RUNNING;
    idle->counter = idle->priority = 0; //由于 idle 不参与调度 可以将其 counter
/ priority 设置为 0
    idle->pid = 0; // 设置 idle 的 pid 为 0
    current = task[0] = idle; //将 current 和 task[0] 指向 idle
    /* YOUR CODE HERE */
    for(int i = 1; i < NR_TASKS; i++){ // 为 task[1] ~ task[NR_TASKS - 1] 进行
初始化
        uint64 task_addr = kalloc(); // 调用 kalloc() 为 task[i] 分配一个物理页
        task[i] = (struct task_struct*)task_addr; // 设置 state 为
TASK_RUNNING;
        task[i]->state = TASK_RUNNING; // 设置 state 为 TASK_RUNNING;
        task[i]->counter = 0; // 设置 counter 为 0;
        task[i]->priority = rand() % 10 + 1; // 利用rand() 设置 priority;
        task[i]->pid = i; // 设置pid，pid 为该线程在线程数组中的下标
        task[i]->thread.ra = (uint64)__dummy;
        task[i]->thread.sp = task_addr + 4096;
        // 为 task[i] 设置 `thread_struct` 中的 `ra` 和 `sp`,
        // 其中 `ra` 设置为 __dummy （见 4.3.2）的地址,  `sp` 设置为 该线程申请的物
理页的高地址
```

```
    }

    printk("...proc_init done!\n");
}
```

4.3.2 `__dummy` 与 `dummy` 介绍

- `task[1]` ~ `task[NR_TASKS - 1]` 都运行同一段代码 `dummy()` 我们在 `proc.c` 添加 `dummy()`:

```
// arch/riscv/kernel/proc.c

void dummy() {
    uint64 MOD = 1000000007;
    uint64 auto_inc_local_var = 0;
    int last_counter = -1;
    while(1) {
        if (last_counter == -1 || current->counter != last_counter) {
            last_counter = current->counter;
            auto_inc_local_var = (auto_inc_local_var + 1) % MOD;
            printk("[PID = %d] is running. auto_inc_local_var = %d\n",
current->pid, auto_inc_local_var);
        }
    }
}
```

> Debug 提示： 可以用 printk 打印更多的信息

- 当线程在运行时, 由于时钟中断的触发, 会将当前运行线程的上下文环境保存在栈上。当线程再次被调度时, 会将上下文从栈上恢复, 但是当我们创建一个新的线程, 此时线程的栈为空, 当这个线程被调度时, 是没有上下文需要被恢复的, 所以我们需要为线程 第一次调度 提供一个特殊的返回函数 `__dummy`
- 在 `entry.S` 添加 `__dummy`
  - 在 `__dummy` 中将 sepc 设置为 `dummy()` 的地址, 并使用 `sret` 从中断中返回。
  - `__dummy` 与 `_traps` 的 `restore` 部分相比, 其实就是省略了从栈上恢复上下文的过程 (但是手动设置了 `sepc`)。

```
# arch/riscv/kernel/entry.S

    .global __dummy
__dummy:
    # YOUR CODE HERE
```

4.3.3 实现线程切换

- 判断下一个执行的线程 `next` 与当前的线程 `current` 是否为同一个线程, 如果是同一个线程, 则无需做任何处理, 否则调用 `__switch_to` 进行线程切换。

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

void switch_to(struct task_struct* next) {
    if (next == current) return; // 当线程转换的目标是本身的线程时，无需进行操作
    else {
        struct task_struct*  temp= current;
        current = next;
        __switch_to(temp, next);
    }
}
```

- 在 `entry.S` 中实现线程上下文切换 `__switch_to`：
  - `__switch_to` 接受两个 `task_struct` 指针作为参数
  - 保存当前线程的 `ra`，`sp`，`s0~s11` 到当前线程的 `thread_struct` 中
  - 将下一个线程的 `thread_struct` 中的相关数据载入到 `ra`，`sp`，`s0~s11` 中。

```asm
    .globl __switch_to
__switch_to:
    # save state to prev process
    # including ra, sp, s0-s11
    add t0, a0, 40  # offset of thread_struct in current task_struct
    sd ra, 0(t0)
    sd sp, 8(t0)
    sd s0, 16(t0)
    sd s1, 24(t0)
    sd s2, 32(t0)
    sd s3, 40(t0)
    sd s4, 48(t0)
    sd s5, 56(t0)
    sd s6, 64(t0)
    sd s7, 72(t0)
    sd s8, 80(t0)
    sd s9, 88(t0)
    sd s10, 96(t0)
    sd s11, 104(t0)

    # restore state from next process
    add t0, a1, 40  # offset of thread_struct in next task_struct
    ld ra, 0(t0)
    ld sp, 8(t0)
    ld s0, 16(t0)
    ld s1, 24(t0)
    ld s2, 32(t0)
    ld s3, 40(t0)
    ld s4, 48(t0)
    ld s5, 56(t0)
    ld s6, 64(t0)
    ld s7, 72(t0)
    ld s8, 80(t0)
    ld s9, 88(t0)
    ld s10, 96(t0)
    ld s11, 104(t0)
```

```
    ret
```

4.3.4 实现调度入口函数

- 实现 `do_timer()`，并在 `时钟中断处理函数` 中调用。

```
void do_timer(void) {
    if (current == task[0]) schedule();
        // 如果当前线程是 idle 线程,直接进行调度,无需其他处理
    else {
        // 如果当前线程不是 idle，则当前线程的运行剩余时间减1；剩余时间大于0则无需调度
        current->counter -= 1;
        if (current->counter == 0) schedule();
    }
}
```

4.3.5 实现线程调度

本次实验我们需要实现两种调度算法：1.短作业优先调度算法, 2.优先级调度算法。

4.3.5.1 短作业优先调度算法

- 当需要进行调度时按照一下规则进行调度：
  - 遍历线程指针数组 `task` (不包括 `idle`，即 `task[0]`), 在所有运行状态 (`TASK_RUNNING`) 下的线程运行剩余时间 `最少` 的线程作为下一个执行的线程。
  - 如果 `所有` 运行状态下的线程运行剩余时间都为0, 则对 `task[1]` ~ `task[NR_TASKS-1]` 的运行剩余时间重新赋值 (使用 `rand()`), 之后再重新进行调度。

```
void schedule(void){

    uint64 min = INF; //  将min设为def.h中定义的最大值
    struct task_struct* next = NULL;
    int flag = 1; // 使用flag来检测目前是否还存在需要调度的进程
    for(int i = 1; i < NR_TASKS; i++){
        if (task[i]->state == TASK_RUNNING && task[i]->counter > 0) { // 遍历
线程指针数组
            flag = 0; // 当还存在需要调度的进程时，flag = 0
            if (task[i]->counter < min) { //找到所有运行状态下的线程中，运行剩余时
间最少的线程，作为下一个执行的线程
                min = task[i]->counter;
                next = task[i];
            }
        }
    }

    if (flag) { // 当目前已经不存在需要调度的进程时，即所有运行状态下的线程运行剩余时间都
为0
        printk("\n");
        for(int i = 1; i < NR_TASKS; i++){
            // 对task[1]~task[NR_TASKS-1]的运行剩余时间重新赋值
            task[i]->counter = rand() % 10 + 1;
            printk("SET [PID = %d COUNTER = %d]\n", task[i]->pid, task[i]-
>counter);
        }
```

```
        // 重新进行调度
        schedule();
    }
    else {
        // 转移到需要执行的进程
        if (next) {
            printk("\nswitch to [PID = %d COUNTER = %d]\n", next->pid, next-
>counter);
            switch_to(next);
        }
    }
}
```

4.3.5.2 优先级调度算法

- 参考 Linux v0.11 调度算法实现 实现。

```
void schedule(void){
    uint64 c, i, next;
    struct task_struct ** p;

    while(1) {
        c = 0;
        next = 0;
        i = NR_TASKS;
        p = &task[NR_TASKS];
        // 遍历任务列表，找到下一个要运行的任务
        while(--i) {
            if (!*--p) continue;
            // 如果任务的状态为运行状态且时间片大于 c，则更新 c 和下一个任务的索引
            if ((*p)->state == TASK_RUNNING && (*p)->counter > c)
                c = (*p)->counter, next = i;
        }

        if (c) break; // 如果找到了下一个任务，跳出循环

        printk("\n");
        // 如果没有找到可运行的任务，进行时间片重分配
        for(p = &task[NR_TASKS-1]; p > &task[0]; --p) {
            if (*p) {
                // 将任务的时间片减少，再加上任务的优先级
                (*p)->counter = ((*p)->counter >> 1) + (*p)->priority;
                printk("SET [PID = %d PRIORITY = %d COUNTER = %d]\n", (*p)-
>pid, (*p)->priority, (*p)->counter);
            }
        }
    }

    printk("\nswitch to [PID = %d PRIORITY = %d COUNTER = %d]\n", task[next]-
>pid, task[next]->priority, task[next]->counter);
    // 切换到下一个任务
    switch_to(task[next]);
}
```

4.4 编译及测试

- 由于加入了一些新的 .c 文件, 可能需要修改一些Makefile文件, 请同学自己尝试修改, 使项目可以编译并运行。
- 由于本次实验需要完成两个调度算法, 因此需要两种调度算法可以使用`gcc -D`选项进行控制。
  - DSJF （短作业优先调度）。
  - DPRIORITY （优先级调度）。
  - 在`proc.c`中使用`#ifdef`, `#endif`来控制代码。 修改顶层Makefile为`CFLAG = ${CF} ${INCLUDE} -DSJF`或`CFLAG = ${CF} ${INCLUDE} -DPRIORITY`(作业提交的时候`Makefile`选择任意一个都可以)
- 短作业优先调度输出结果

```
OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : timer,mfdeleg
Platform HART Count       : 1
Firmware Base             : 0x80000000
Firmware Size             : 100 KB
Runtime SBI Version       : 0.2

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*
Domain0 Region00          : 0x0000000080000000-0x000000008001ffff ()
Domain0 Region01          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000080200000
Domain0 Next Arg1         : 0x0000000087000000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
SET [PID = 13 COUNTER = 1]
SET [PID = 14 COUNTER = 3]
SET [PID = 15 COUNTER = 5]
SET [PID = 16 COUNTER = 1]
SET [PID = 17 COUNTER = 1]
SET [PID = 18 COUNTER = 6]
SET [PID = 19 COUNTER = 8]
SET [PID = 20 COUNTER = 9]
SET [PID = 21 COUNTER = 5]
SET [PID = 22 COUNTER = 9]
SET [PID = 23 COUNTER = 8]
SET [PID = 24 COUNTER = 7]
SET [PID = 25 COUNTER = 6]
SET [PID = 26 COUNTER = 7]
SET [PID = 27 COUNTER = 1]
SET [PID = 28 COUNTER = 2]
SET [PID = 29 COUNTER = 4]
SET [PID = 30 COUNTER = 9]
SET [PID = 31 COUNTER = 9]
```

```
switch to [PID = 6 COUNTER = 1]
[PID = 6] is running. auto_inc_local_var = 1

switch to [PID = 8 COUNTER = 1]
[PID = 8] is running. auto_inc_local_var = 1

switch to [PID = 13 COUNTER = 1]
[PID = 13] is running. auto_inc_local_var = 1

switch to [PID = 16 COUNTER = 1]
[PID = 16] is running. auto_inc_local_var = 1

switch to [PID = 17 COUNTER = 1]
[PID = 17] is running. auto_inc_local_var = 1

switch to [PID = 27 COUNTER = 1]
[PID = 27] is running. auto_inc_local_var = 1

switch to [PID = 12 COUNTER = 2]
[PID = 12] is running. auto_inc_local_var = 1
[PID = 12] is running. auto_inc_local_var = 2

switch to [PID = 28 COUNTER = 2]
[PID = 28] is running. auto_inc_local_var = 1
[PID = 28] is running. auto_inc_local_var = 2

switch to [PID = 1 COUNTER = 3]
[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

switch to [PID = 2 COUNTER = 3]
[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

switch to [PID = 9 COUNTER = 3]
[PID = 9] is running. auto_inc_local_var = 1
[PID = 9] is running. auto_inc_local_var = 2
[PID = 9] is running. auto_inc_local_var = 3

switch to [PID = 14 COUNTER = 3]
[PID = 14] is running. auto_inc_local_var = 1
[PID = 14] is running. auto_inc_local_var = 2
[PID = 14] is running. auto_inc_local_var = 3

switch to [PID = 11 COUNTER = 4]
[PID = 11] is running. auto_inc_local_var = 1
[PID = 11] is running. auto_inc_local_var = 2
[PID = 11] is running. auto_inc_local_var = 3
[PID = 11] is running. auto_inc_local_var = 4
```

```
switch to [PID = 29 COUNTER = 4]
[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

switch to [PID = 15 COUNTER = 5]
[PID = 15] is running. auto_inc_local_var = 1
[PID = 15] is running. auto_inc_local_var = 2
[PID = 15] is running. auto_inc_local_var = 3
[PID = 15] is running. auto_inc_local_var = 4
[PID = 15] is running. auto_inc_local_var = 5

switch to [PID = 21 COUNTER = 5]
[PID = 21] is running. auto_inc_local_var = 1
[PID = 21] is running. auto_inc_local_var = 2
[PID = 21] is running. auto_inc_local_var = 3
[PID = 21] is running. auto_inc_local_var = 4
[PID = 21] is running. auto_inc_local_var = 5

switch to [PID = 4 COUNTER = 6]
[PID = 4] is running. auto_inc_local_var = 1
[PID = 4] is running. auto_inc_local_var = 2
[PID = 4] is running. auto_inc_local_var = 3
[PID = 4] is running. auto_inc_local_var = 4
[PID = 4] is running. auto_inc_local_var = 5
[PID = 4] is running. auto_inc_local_var = 6

switch to [PID = 5 COUNTER = 6]
[PID = 5] is running. auto_inc_local_var = 1
[PID = 5] is running. auto_inc_local_var = 2
[PID = 5] is running. auto_inc_local_var = 3
[PID = 5] is running. auto_inc_local_var = 4
[PID = 5] is running. auto_inc_local_var = 5
[PID = 5] is running. auto_inc_local_var = 6

switch to [PID = 18 COUNTER = 6]
[PID = 18] is running. auto_inc_local_var = 1
[PID = 18] is running. auto_inc_local_var = 2
[PID = 18] is running. auto_inc_local_var = 3
[PID = 18] is running. auto_inc_local_var = 4
[PID = 18] is running. auto_inc_local_var = 5
[PID = 18] is running. auto_inc_local_var = 6

switch to [PID = 25 COUNTER = 6]
[PID = 25] is running. auto_inc_local_var = 1
[PID = 25] is running. auto_inc_local_var = 2
[PID = 25] is running. auto_inc_local_var = 3
[PID = 25] is running. auto_inc_local_var = 4
[PID = 25] is running. auto_inc_local_var = 5
[PID = 25] is running. auto_inc_local_var = 6
```

```
switch to [PID = 10 COUNTER = 7]
[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

switch to [PID = 24 COUNTER = 7]
[PID = 24] is running. auto_inc_local_var = 1
[PID = 24] is running. auto_inc_local_var = 2
[PID = 24] is running. auto_inc_local_var = 3
[PID = 24] is running. auto_inc_local_var = 4
[PID = 24] is running. auto_inc_local_var = 5
[PID = 24] is running. auto_inc_local_var = 6
[PID = 24] is running. auto_inc_local_var = 7

switch to [PID = 26 COUNTER = 7]
[PID = 26] is running. auto_inc_local_var = 1
[PID = 26] is running. auto_inc_local_var = 2
[PID = 26] is running. auto_inc_local_var = 3
[PID = 26] is running. auto_inc_local_var = 4
[PID = 26] is running. auto_inc_local_var = 5
[PID = 26] is running. auto_inc_local_var = 6
[PID = 26] is running. auto_inc_local_var = 7

switch to [PID = 3 COUNTER = 8]
[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
[PID = 3] is running. auto_inc_local_var = 5
[PID = 3] is running. auto_inc_local_var = 6
[PID = 3] is running. auto_inc_local_var = 7
[PID = 3] is running. auto_inc_local_var = 8

switch to [PID = 19 COUNTER = 8]
[PID = 19] is running. auto_inc_local_var = 1
[PID = 19] is running. auto_inc_local_var = 2
[PID = 19] is running. auto_inc_local_var = 3
[PID = 19] is running. auto_inc_local_var = 4
[PID = 19] is running. auto_inc_local_var = 5
[PID = 19] is running. auto_inc_local_var = 6
[PID = 19] is running. auto_inc_local_var = 7
[PID = 19] is running. auto_inc_local_var = 8
```

```
[PID = 23] is running. auto_inc_local_var = 8

switch to [PID = 20 COUNTER = 9]
[PID = 20] is running. auto_inc_local_var = 1
[PID = 20] is running. auto_inc_local_var = 2
[PID = 20] is running. auto_inc_local_var = 3
[PID = 20] is running. auto_inc_local_var = 4
[PID = 20] is running. auto_inc_local_var = 5
[PID = 20] is running. auto_inc_local_var = 6
[PID = 20] is running. auto_inc_local_var = 7
[PID = 20] is running. auto_inc_local_var = 8
[PID = 20] is running. auto_inc_local_var = 9

switch to [PID = 22 COUNTER = 9]
[PID = 22] is running. auto_inc_local_var = 1
[PID = 22] is running. auto_inc_local_var = 2
[PID = 22] is running. auto_inc_local_var = 3
[PID = 22] is running. auto_inc_local_var = 4
[PID = 22] is running. auto_inc_local_var = 5
[PID = 22] is running. auto_inc_local_var = 6
[PID = 22] is running. auto_inc_local_var = 7
[PID = 22] is running. auto_inc_local_var = 8
[PID = 22] is running. auto_inc_local_var = 9

switch to [PID = 30 COUNTER = 9]
[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

switch to [PID = 31 COUNTER = 9]
[PID = 31] is running. auto_inc_local_var = 1
[PID = 31] is running. auto_inc_local_var = 2
[PID = 31] is running. auto_inc_local_var = 3
[PID = 31] is running. auto_inc_local_var = 4
[PID = 31] is running. auto_inc_local_var = 5
[PID = 31] is running. auto_inc_local_var = 6
[PID = 31] is running. auto_inc_local_var = 7
[PID = 31] is running. auto_inc_local_var = 8
[PID = 31] is running. auto_inc_local_var = 9

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

- 优先级调度输出结果

```
OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : timer,mfdeleg
Platform HART Count       : 1
Firmware Base             : 0x80000000
Firmware Size             : 100 KB
Runtime SBI Version       : 0.2

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*
Domain0 Region00          : 0x0000000080000000-0x0000000080001fff ()
Domain0 Region01          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000080200000
Domain0 Next Arg1         : 0x0000000087000000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain          : root
Boot HART ISA             : rv64imafdcsu
Boot HART Features        : scounteren,mcounteren,time
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 0
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
...mm_init done!
...proc_init done!
2023 Hello RISC-V
idle process is running!

SET [PID = 31 PRIORITY = 5 COUNTER = 5]
SET [PID = 30 PRIORITY = 10 COUNTER = 10]
SET [PID = 29 PRIORITY = 9 COUNTER = 9]
SET [PID = 28 PRIORITY = 4 COUNTER = 4]
SET [PID = 27 PRIORITY = 2 COUNTER = 2]
SET [PID = 26 PRIORITY = 1 COUNTER = 1]
SET [PID = 25 PRIORITY = 9 COUNTER = 9]
SET [PID = 24 PRIORITY = 4 COUNTER = 4]
SET [PID = 23 PRIORITY = 1 COUNTER = 1]
```

```
SET [PID = 21 PRIORITY = 7 COUNTER = 7]
SET [PID = 20 PRIORITY = 10 COUNTER = 10]
SET [PID = 19 PRIORITY = 9 COUNTER = 9]
SET [PID = 18 PRIORITY = 9 COUNTER = 9]
SET [PID = 17 PRIORITY = 6 COUNTER = 6]
SET [PID = 16 PRIORITY = 8 COUNTER = 8]
SET [PID = 15 PRIORITY = 5 COUNTER = 5]
SET [PID = 14 PRIORITY = 1 COUNTER = 1]
SET [PID = 13 PRIORITY = 6 COUNTER = 6]
SET [PID = 12 PRIORITY = 1 COUNTER = 1]
SET [PID = 11 PRIORITY = 5 COUNTER = 5]
SET [PID = 10 PRIORITY = 5 COUNTER = 5]
SET [PID = 9 PRIORITY = 10 COUNTER = 10]
SET [PID = 8 PRIORITY = 3 COUNTER = 3]
SET [PID = 7 PRIORITY = 6 COUNTER = 6]
SET [PID = 6 PRIORITY = 1 COUNTER = 1]
SET [PID = 5 PRIORITY = 1 COUNTER = 1]
SET [PID = 4 PRIORITY = 5 COUNTER = 5]
SET [PID = 3 PRIORITY = 1 COUNTER = 1]
SET [PID = 2 PRIORITY = 5 COUNTER = 5]
SET [PID = 1 PRIORITY = 2 COUNTER = 2]

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 = 9] is running. auto_inc_local_var = 4
[PID = 9] is running. auto_inc_local_var = 5
[PID = 9] is running. auto_inc_local_var = 6
[PID = 9] is running. auto_inc_local_var = 7
[PID = 9] is running. auto_inc_local_var = 8
[PID = 9] is running. auto_inc_local_var = 9
[PID = 9] is running. auto_inc_local_var = 10

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 = 25 PRIORITY = 9 COUNTER = 9]
[PID = 25] is running. auto_inc_local_var = 1
[PID = 25] is running. auto_inc_local_var = 2
[PID = 25] is running. auto_inc_local_var = 3
[PID = 25] is running. auto_inc_local_var = 4
[PID = 25] is running. auto_inc_local_var = 5
[PID = 25] is running. auto_inc_local_var = 6
[PID = 25] is running. auto_inc_local_var = 7
[PID = 25] is running. auto_inc_local_var = 8
[PID = 25] is running. auto_inc_local_var = 9

switch to [PID = 22 PRIORITY = 9 COUNTER = 9]
[PID = 22] is running. auto_inc_local_var = 1
[PID = 22] is running. auto_inc_local_var = 2
[PID = 22] is running. auto_inc_local_var = 3
[PID = 22] is running. auto_inc_local_var = 4
[PID = 22] is running. auto_inc_local_var = 5
[PID = 22] is running. auto_inc_local_var = 6
[PID = 22] is running. auto_inc_local_var = 7
[PID = 22] is running. auto_inc_local_var = 8
[PID = 22] is running. auto_inc_local_var = 9

switch to [PID = 19 PRIORITY = 9 COUNTER = 9]
[PID = 19] is running. auto_inc_local_var = 1
[PID = 19] is running. auto_inc_local_var = 2
[PID = 19] is running. auto_inc_local_var = 3
[PID = 19] is running. auto_inc_local_var = 4
[PID = 19] is running. auto_inc_local_var = 5
[PID = 19] is running. auto_inc_local_var = 6
[PID = 19] is running. auto_inc_local_var = 7
[PID = 19] is running. auto_inc_local_var = 8
[PID = 19] is running. auto_inc_local_var = 9

switch to [PID = 18 PRIORITY = 9 COUNTER = 9]
[PID = 18] is running. auto_inc_local_var = 1
[PID = 18] is running. auto_inc_local_var = 2
[PID = 18] is running. auto_inc_local_var = 3
[PID = 18] is running. auto_inc_local_var = 4
[PID = 18] is running. auto_inc_local_var = 5
[PID = 18] is running. auto_inc_local_var = 6
[PID = 18] is running. auto_inc_local_var = 7
[PID = 18] is running. auto_inc_local_var = 8
[PID = 18] 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
```

```
switch to [PID = 21 PRIORITY = 7 COUNTER = 7]
[PID = 21] is running. auto_inc_local_var = 1
[PID = 21] is running. auto_inc_local_var = 2
[PID = 21] is running. auto_inc_local_var = 3
[PID = 21] is running. auto_inc_local_var = 4
[PID = 21] is running. auto_inc_local_var = 5
[PID = 21] is running. auto_inc_local_var = 6
[PID = 21] is running. auto_inc_local_var = 7

switch to [PID = 17 PRIORITY = 6 COUNTER = 6]
[PID = 17] is running. auto_inc_local_var = 1
[PID = 17] is running. auto_inc_local_var = 2
[PID = 17] is running. auto_inc_local_var = 3
[PID = 17] is running. auto_inc_local_var = 4
[PID = 17] is running. auto_inc_local_var = 5
[PID = 17] is running. auto_inc_local_var = 6

switch to [PID = 13 PRIORITY = 6 COUNTER = 6]
[PID = 13] is running. auto_inc_local_var = 1
[PID = 13] is running. auto_inc_local_var = 2
[PID = 13] is running. auto_inc_local_var = 3
[PID = 13] is running. auto_inc_local_var = 4
[PID = 13] is running. auto_inc_local_var = 5
[PID = 13] is running. auto_inc_local_var = 6

switch to [PID = 7 PRIORITY = 6 COUNTER = 6]
[PID = 7] is running. auto_inc_local_var = 1
[PID = 7] is running. auto_inc_local_var = 2
[PID = 7] is running. auto_inc_local_var = 3
[PID = 7] is running. auto_inc_local_var = 4
[PID = 7] is running. auto_inc_local_var = 5
[PID = 7] is running. auto_inc_local_var = 6

switch to [PID = 31 PRIORITY = 5 COUNTER = 5]
[PID = 31] is running. auto_inc_local_var = 1
[PID = 31] is running. auto_inc_local_var = 2
[PID = 31] is running. auto_inc_local_var = 3
[PID = 31] is running. auto_inc_local_var = 4
[PID = 31] is running. auto_inc_local_var = 5

switch to [PID = 15 PRIORITY = 5 COUNTER = 5]
[PID = 15] is running. auto_inc_local_var = 1
[PID = 15] is running. auto_inc_local_var = 2
[PID = 15] is running. auto_inc_local_var = 3
[PID = 15] is running. auto_inc_local_var = 4
[PID = 15] is running. auto_inc_local_var = 5

switch to [PID = 11 PRIORITY = 5 COUNTER = 5]
```

```
switch to [PID = 10 PRIORITY = 5 COUNTER = 5]
[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

switch to [PID = 4 PRIORITY = 5 COUNTER = 5]
[PID = 4] is running. auto_inc_local_var = 1
[PID = 4] is running. auto_inc_local_var = 2
[PID = 4] is running. auto_inc_local_var = 3
[PID = 4] is running. auto_inc_local_var = 4
[PID = 4] is running. auto_inc_local_var = 5

switch to [PID = 2 PRIORITY = 5 COUNTER = 5]
[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

switch to [PID = 28 PRIORITY = 4 COUNTER = 4]
[PID = 28] is running. auto_inc_local_var = 1
[PID = 28] is running. auto_inc_local_var = 2
[PID = 28] is running. auto_inc_local_var = 3
[PID = 28] is running. auto_inc_local_var = 4

switch to [PID = 24 PRIORITY = 4 COUNTER = 4]
[PID = 24] is running. auto_inc_local_var = 1
[PID = 24] is running. auto_inc_local_var = 2
[PID = 24] is running. auto_inc_local_var = 3
[PID = 24] is running. auto_inc_local_var = 4

switch to [PID = 8 PRIORITY = 3 COUNTER = 3]
[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

switch to [PID = 27 PRIORITY = 2 COUNTER = 2]
[PID = 27] is running. auto_inc_local_var = 1
[PID = 27] is running. auto_inc_local_var = 2

switch to [PID = 1 PRIORITY = 2 COUNTER = 2]
[PID = 1] is running. auto_inc_local_var = 1
[PID = 1] is running. auto_inc_local_var = 2

switch to [PID = 26 PRIORITY = 1 COUNTER = 1]
[PID = 26] is running. auto_inc_local_var = 1
```

思考题

```
switch to [PID = 23 PRIORITY = 1 COUNTER = 1]
[PID = 23] is running. auto_inc_local_var = 1
```

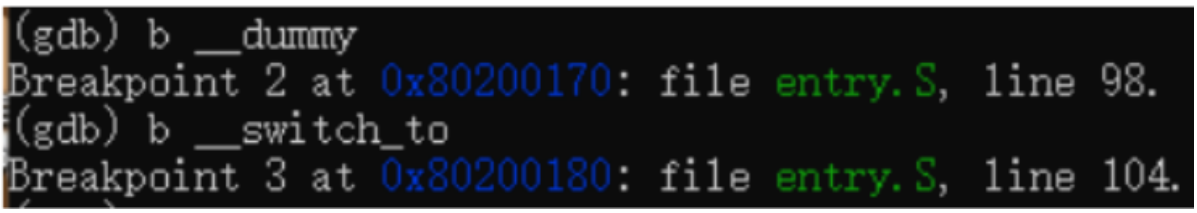
1. **在 RV64 中一共用 32 个通用寄存器，为什么 `context_switch` 中只保存了14个？**

   因为实验中实现的线程切换是从 C 语言代码调用 `__switch_to` 函数，而 Caller Saved Register 会被 C 编译器保存在当前的栈上，因此在 `__switch_to` 函数只需要处理 C 编译器不会保存，但是 `__switch_to` 函数有需要的一些寄存器。所以在线程切换时只需要保存所有 callee-saved register 和 `ra` 以及 `sp`。

2. **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 `context_switch` 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)。**

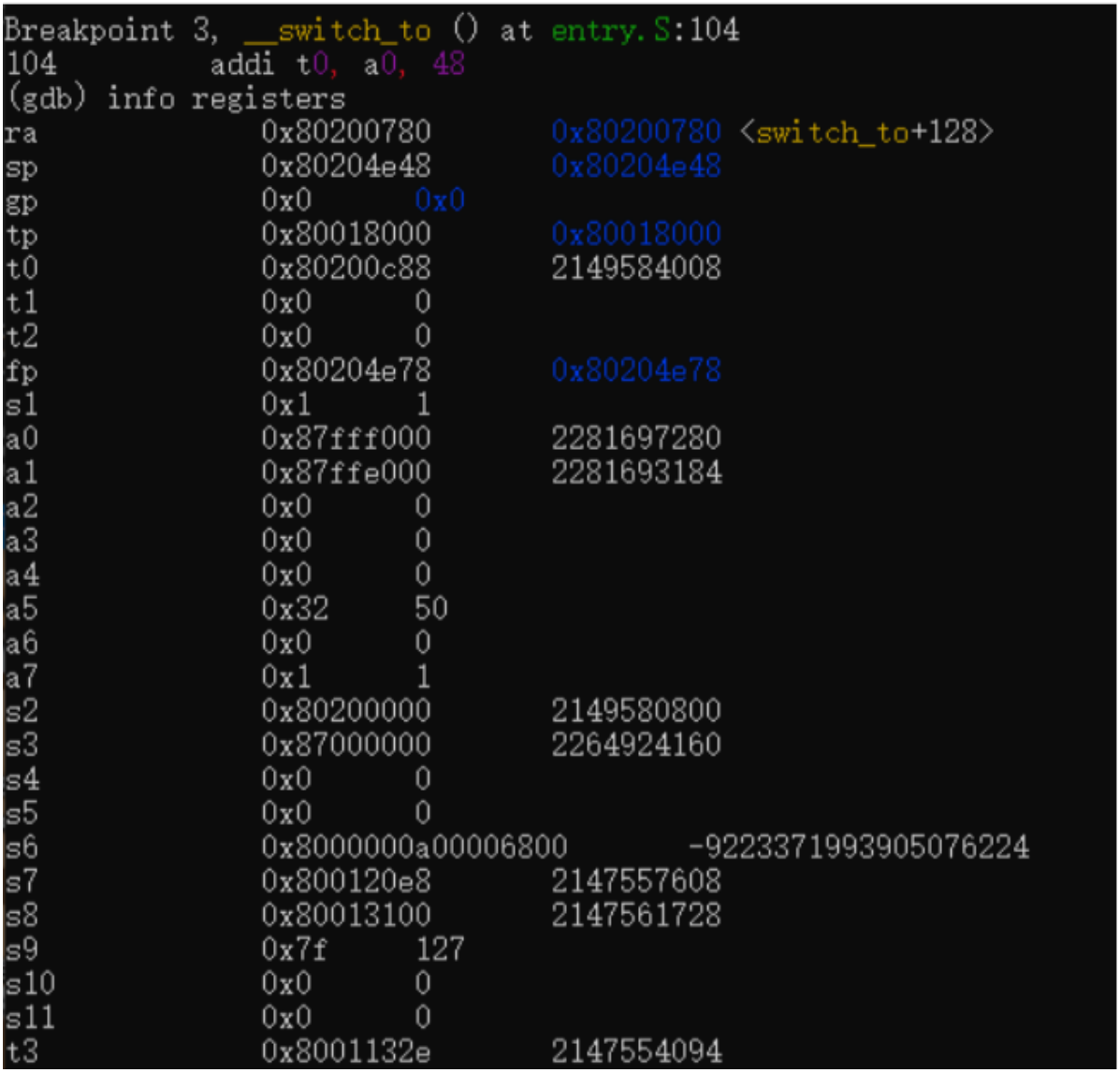
   保存恢复的函数返回点也是 __dummy，经过 __dummy 设置返回地址__dummy( ) 从中断回到 __dummy( ) 的地址继续执行，此时的寄存器内容已经是切换后的线程的内容。
   在 start_kernel之后，ra 的值为__traps 的地址

在 __dummy 和 __switch_to 设置断点：



由系统进程切换到第一个进程：

切换前：

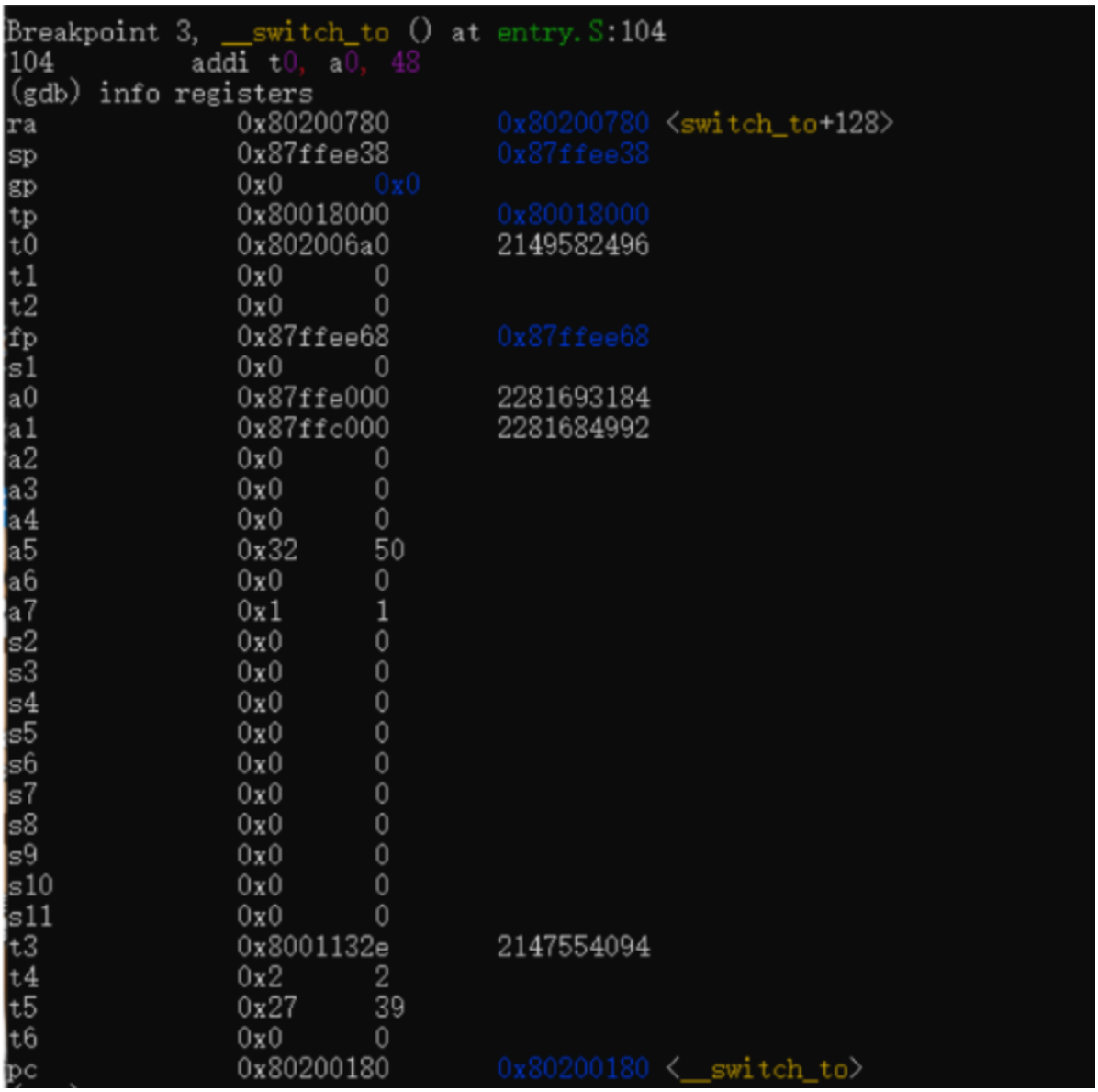

Breakpoint 3, __switch_to () at entry.S:104
104 addi t0, a0, 48
(gdb) info registers
ra 0x80200780 0x80200780 <switch_to+128>
sp 0x80204e48 0x80204e48
gp 0x0 0x0
tp 0x80018000 0x80018000
t0 0x80200c88 2149584008
t1 0x0 0
t2 0x0 0
fp 0x80204e78 0x80204e78
s1 0x1 1
a0 0x87fff000 2281697280
a1 0x87ffe000 2281693184
a2 0x0 0
a3 0x0 0
a4 0x0 0
a5 0x32 50
a6 0x0 0
a7 0x1 1
s2 0x80200000 2149580800
s3 0x87000000 2264924160
s4 0x0 0
s5 0x0 0
s6 0x8000000a00006800 -9223371993905076224
s7 0x800120e8 2147557608
s8 0x80013100 2147561728
s9 0x7f 127
s10 0x0 0
s11 0x0 0
t3 0x8001132e 2147554094

切换后:

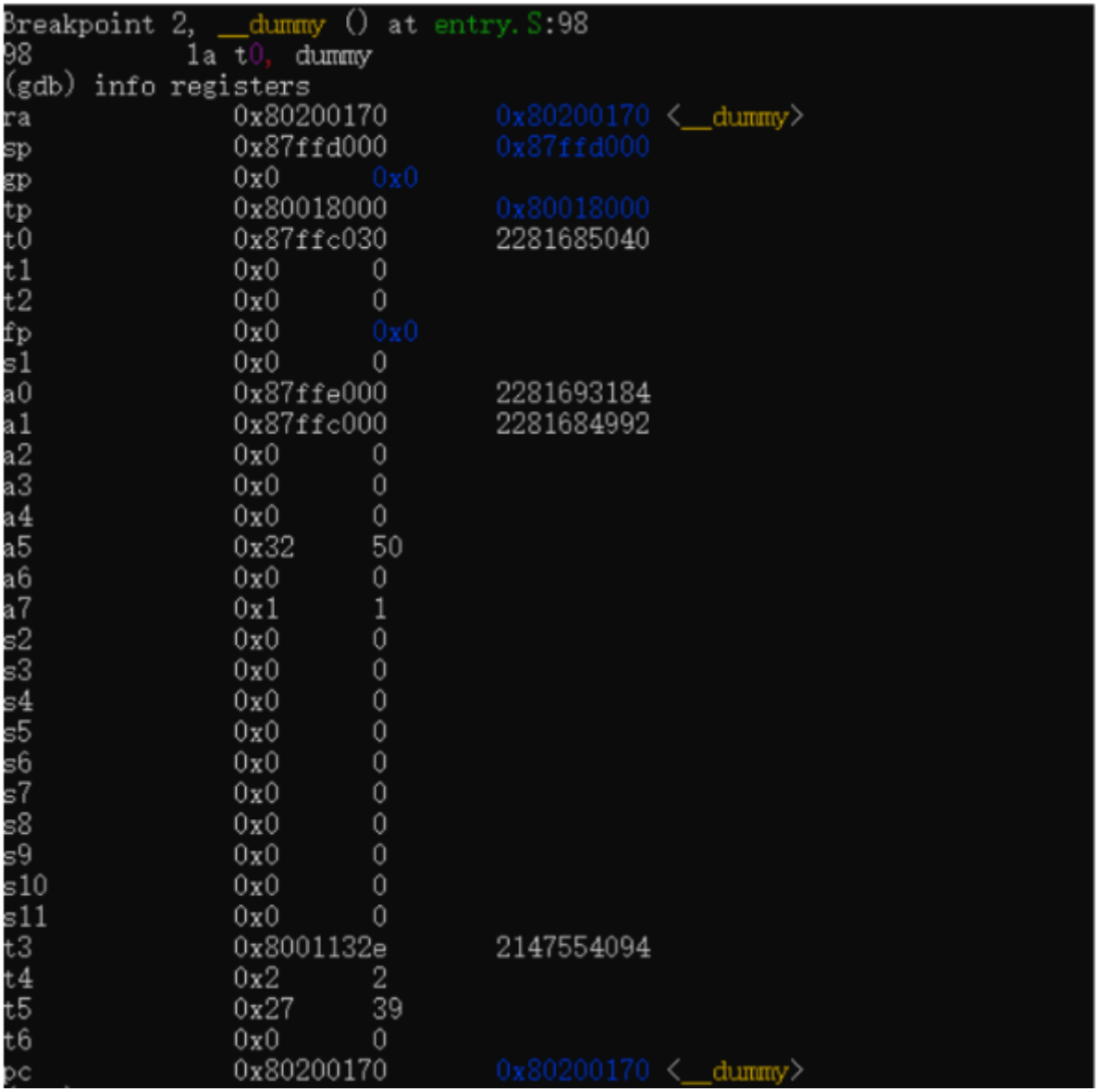
```
Breakpoint 2, __dummy () at entry.S:98
98          la t0, dummy
(gdb) info registers
ra             0x80200170       0x80200170 <__dummy>
sp             0x87fff000       0x87fff000
gp             0x0      0x0
tp             0x80018000       0x80018000
t0             0x87ffe030       2281693232
t1             0x0      0
t2             0x0      0
fp             0x0      0x0
s1             0x0      0
a0             0x87fff000       2281697280
a1             0x87ffe000       2281693184
a2             0x0      0
a3             0x0      0
a4             0x0      0
a5             0x32     50
a6             0x0      0
a7             0x1      1
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             0x8001132e       2147554094
t4             0x2      2
t5             0x27     39
t6             0x0      0
pc             0x80200170       0x80200170 <__dummy>
```

由第一个进程切换到第二个进程：

切换前：


Breakpoint 3, __switch_to () at entry.S:104
104 addi t0, a0, 48
(gdb) info registers
ra 0x80200780 0x80200780 <switch_to+128>
sp 0x87ffee38 0x87ffee38
gp 0x0 0x0
tp 0x80018000 0x80018000
t0 0x802006a0 2149582496
t1 0x0 0
t2 0x0 0
fp 0x87ffee68 0x87ffee68
s1 0x0 0
a0 0x87ffe000 2281693184
a1 0x87ffc000 2281684992
a2 0x0 0
a3 0x0 0
a4 0x0 0
a5 0x32 50
a6 0x0 0
a7 0x1 1
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 0x8001132e 2147554094
t4 0x2 2
t5 0x27 39
t6 0x0 0
pc 0x80200180 0x80200180 <__switch_to>

切换后:


Breakpoint 2, __dummy () at entry.S:98
98 la t0, dummy
(gdb) info registers
ra 0x80200170 0x80200170 <__dummy>
sp 0x87ffd000 0x87ffd000
gp 0x0 0x0
tp 0x80018000 0x80018000
t0 0x87ffc030 2281685040
t1 0x0 0
t2 0x0 0
fp 0x0 0x0
s1 0x0 0
a0 0x87ffe000 2281693184
a1 0x87ffc000 2281684992
a2 0x0 0
a3 0x0 0
a4 0x0 0
a5 0x32 50
a6 0x0 0
a7 0x1 1
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 0x8001132e 2147554094
t4 0x2 2
t5 0x27 39
t6 0x0 0
pc 0x80200170 0x80200170 <__dummy>