浙江大学操作系统实验报告

实验名称：Lab 2: RV64 内核线程调度

电子邮件地址：jjhuang535@outlook.com

手机：18972039196

实验地点：曹西503

实验日期：2023.10.30

学号：3210105944

姓名：黄锦骏

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 实验步骤

3.1 准备工程

- 此次实验基于 lab1 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`, `proc.h/proc.c`, `test.h/test_schedule.h`, `schedule_null.c/schedule_test.c` 以及新增的一些 Makefile 的变化。并按照以下步骤将这些文件正确放置。
  - `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
  - `proc.h/proc.c` 是本次实验需要关注的重点
  - `test.h/test_schedule.h` 提供了本次实验单元测试的测试接口
  - `schedule_null.c/schedule_test.c` 提供了在 “加测试 / 不加测试” 两种情况下测试接口的代码实例

```
.
├── arch
|   └── riscv
|       ├── include
|       |   └── mm.h
|       |   └── proc.h
```

```
|       └── kernel
|           └── mm.c
|           └── proc.c
├── include
|   ├── rand.h
|   ├── string.h
|   ├── test.h
|   └── schedule_test.h
|
├── test
|   ├── schedule_null.c
|   ├── schedule_test.c
|   └── Makefile
|
├── lib
|   ├── rand.c
|   └── string.c
|
└── Makefile
```

- 在 lab2 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`

3.2 `proc.h` 数据结构定义

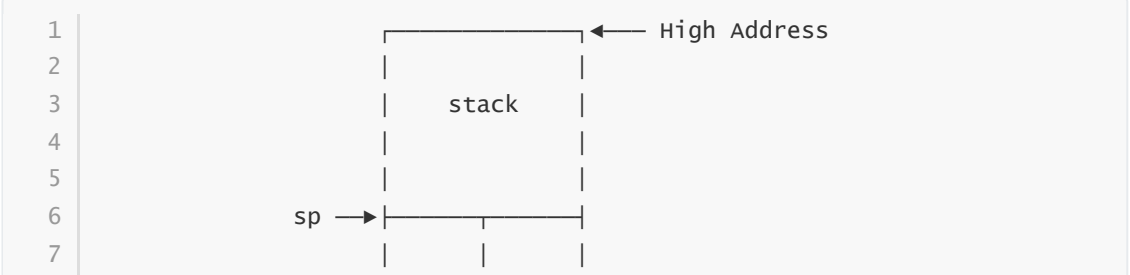
```
// arch/riscv/include/proc.h

#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

```

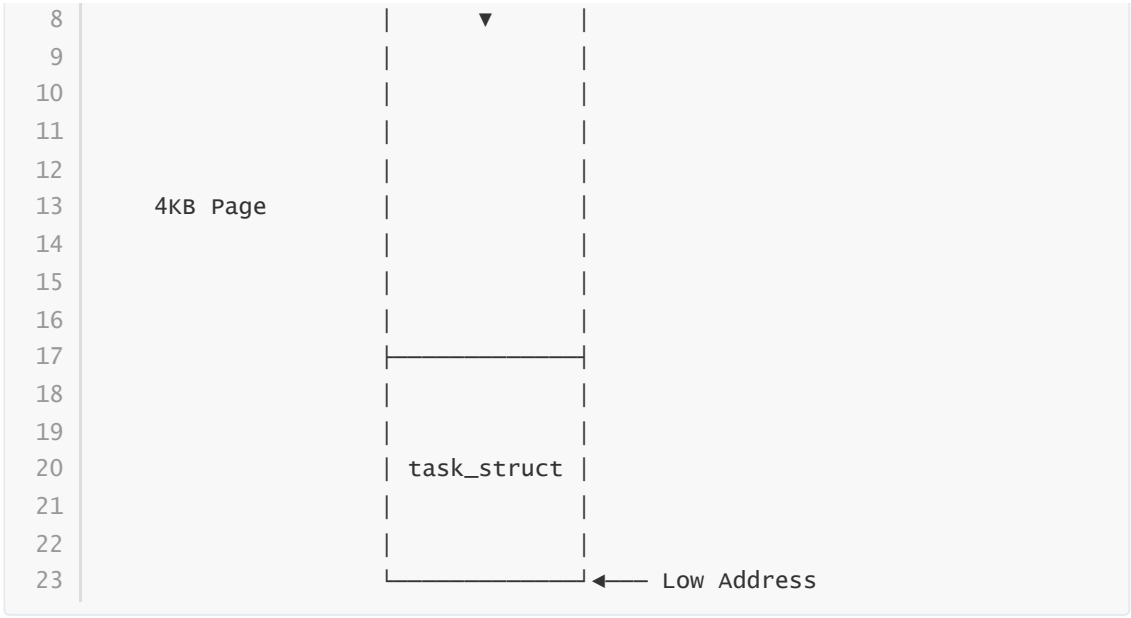
```c
/* 用于记录 `线程` 的 `内核栈与用户栈指针` */
/* (lab3中无需考虑, 在这里引入是为了之后实验的使用) */
struct thread_info {
    uint64 kernel_sp;
    uint64 user_sp;
};

/* 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();
```

3.3 线程调度功能实现

3.3.1 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:





- 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`

最后实现的task_init函数如下所示:

```
void task_init() {
    test_init(NR_TASKS);
    // 1. 调用 kalloc() 为 idle 分配一个物理页
    idle = (struct task_struct *)kalloc();
    // 2. 设置 state 为 TASK_RUNNING;
    idle->state = TASK_RUNNING;
    // 3. 由于 idle 不参与调度 可以将其 counter / priority 设置为 0
    idle->counter = 0;
    idle->priority = 0;
    // 4. 设置 idle 的 pid 为 0
    idle->pid = 0;
    // 5. 将 current 和 task[0] 指向 idle
    task[0] = idle;
    current = idle;
    /* YOUR CODE HERE */

    // 1. 参考 idle 的设置, 为 task[1] ~ task[NR_TASKS - 1] 进行初始化
    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 = task_test_counter[i];
        task[i]->priority = task_test_priority[i];
        // printk("task[i]->pid: %d\n", task[i]->pid);
        // printk("task[i]->counter: %d\n", task[i]->counter);
        // printk("task[i]->priority: %d\n", task[i]->priority);

        task[i]->thread.ra = (uint64)__dummy;
```

```
        task[i]->thread.sp = (uint64)task[i] + 4096;

    }
    // 2. 其中每个线程的 state 为 TASK_RUNNING, 此外，为了单元测试的需要，counter 和 priority 进行如下赋值：
    //      task[i].counter  = task_test_counter[i];
    //      task[i].priority = task_test_priority[i];
    // 3. 为 task[1] ~ task[NR_TASKS - 1] 设置 `thread_struct` 中的 `ra` 和 `sp`，
    // 4. 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");
}
```

3.3.2 `__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() {
    schedule_test();
    uint64 MOD = 1000000007;
    uint64 auto_inc_local_var = 0;
    int last_counter = -1;
    while(1) {
        if ((last_counter == -1 || current->counter != last_counter) && current->counter > 0) {
            if(current->counter == 1){
                --(current->counter);   // forced the counter to be zero if this thread is going to be scheduled
            }                           // in case that the new counter is also 1, leading the information not printed.
            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`)。

添加的 `__dummy` 函数如下所示

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

    .global __dummy
__dummy:
    la t0, dummy
    csrw sepc, t0
    sret
```

3.3.3 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。

  实现的 `switch_to` 如下所示

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

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


void switch_to(struct task_struct* next) {
    if(next->pid == current->pid)
        return;
    // printk("current_pid: %d\n", current->pid);
    // printk("Next_pid: %d\n", next->pid);

    //由于__switch_to只实现了上下文切换功能，所以这里需要对current指针进行更新
    struct task_struct* prev = current;
    current = next;
        // 调用 __switch_to 进行线程切换
    __switch_to(prev, next);
}
```

- 在 `entry.S` 中实现线程上下文切换 :
  - `__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。

  实现的 `__switch_to` 如下所示

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

    .globl __switch_to
__switch_to:
    # task_struct = thread_info + state + counter + priority + pid + thread_info
    # thread initial address: tast_struct + 4 * 8 + 2 * 8 = prev + 48
    sd ra, 48(a0)
```

```
    sd sp, 56(a0)
    sd s0, 64(a0)
    sd s1, 72(a0)
    sd s2, 80(a0)
    sd s3, 88(a0)
    sd s4, 96(a0)
    sd s5, 104(a0)
    sd s6, 112(a0)
    sd s7, 120(a0)
    sd s8, 128(a0)
    sd s9, 136(a0)
    sd s10, 144(a0)
    sd s11, 152(a0)

    ld ra, 48(a1)
    ld sp, 56(a1)
    ld s0, 64(a1)
    ld s1, 72(a1)
    ld s2, 80(a1)
    ld s3, 88(a1)
    ld s4, 96(a1)
    ld s5, 104(a1)
    ld s6, 112(a1)
    ld s7, 120(a1)
    ld s8, 128(a1)
    ld s9, 136(a1)
    ld s10, 144(a1)
    ld s11, 152(a1)

    ret
```

> Debug 提示： 可以尝试是否可以从 idle 正确切换到 process 1

3.3.4 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。

实现的`do_timer()`函数如下

```c
void do_timer() {
    if(current->pid == 0)
        schedule();
    else{
        //  printk("current_pid: %d\n", current->pid);
        //  printk("current_counter: %d\n", current->counter);
        current->counter--;
        if(current->counter == 0)
            schedule();
        else{
            return;
        }
    }
}
```

3.3.5 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。

3.3.5.1 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。

实现的短作业优先调度算法如下：

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

void schedule(void) {
    int min_i = 0, cnt = 0;
    while(1){
        int min_counter = 1000;
        min_i = 1, cnt = 0;
        for(int i = 1; i < NR_TASKS; i++)
        {
            if(task[i]->counter == 0)
                cnt++;
            if(task[i]->state == TASK_RUNNING && task[i]->counter > 0 &&
task[i]->counter < min_counter)
            {
                min_counter = task[i]->counter;
                min_i = i;
            }
        }
        if(cnt == NR_TASKS - 1)
        {
            for(int i = 1; i < NR_TASKS; i++)
                task[i]->counter = rand() % 10;
        }
        else
            break;
    }
    // printk("schedule_Next_pid: %d\n", task[min_i]->pid);
    if(min_i != 0)
        switch_to(task[min_i]);
    else
        printk("Error occured when schedule!\n");
}
```

> Debug 提示：可以先将`NR_TASKS`改为较小的值, 调用`printk`将所有线程的信息打印出来。

3.3.5.2 优先级调度算法

- 参考 Linux v0.11 调度算法实现 实现。

本人实现如下所示：

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

void schedule(void) {
    int c, next, i;
    while(1){
        // printk("Why you are stunned here\n");
        c = 0;
        next = 0;
        i = NR_TASKS;
        struct task_struct ** p = &task[NR_TASKS];
        while(--i > 0){
            // printk("pid: %d\n", task[i]->pid);
            // printk("state: %d\n", task[i]->state);
            // printk("counter: %d\n", task[i]->counter);
            if( task[i]->state == TASK_RUNNING && task[i]->counter > c)
            {
                c = (task[i])->counter;
                next = i;
            }
        }
        // printk("c: %d\n", c);
        if(c > 0) break;
        for(int j = 1; j <= NR_TASKS; j++)
            if (task[j])
                task[j]->counter = (task[j]->counter >> 1) + task[j]->priority;
    }
    // printk("next :%d\n", next);
    if(next != current->pid && next != 0)
        switch_to(task[next]);
}
```

4.4 编译及测试

- 短作业优先调度算法输出测试结果：`#define NR_TASTS(1+15)`

```
Launch the qemu ......

OpenSBI v1.2
   ____                   _____ ____ _____
  / __ \                 / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : medeleg
Platform HART Count       : 1
Platform IPI Device       : aclint-mswi
Platform Timer Device     : aclint-mtimer @ 10000000Hz
Platform Console Device   : uart8250
Platform HSM Device       : ---
```

```
Platform PMU Device       : ---
Platform Reboot Device    : sifive_test
Platform Shutdown Device  : sifive_test
Firmware Base             : 0x80000000
Firmware Size             : 212 KB
Runtime SBI Version       : 1.0

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*
Domain0 Region00          : 0x0000000002000000-0x000000000200ffff (I)
Domain0 Region01          : 0x0000000080000000-0x000000008003ffff ()
Domain0 Region02          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000080200000
Domain0 Next Arg1         : 0x0000000087e00000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain          : root
Boot HART Priv Version    : v1.12
Boot HART Base ISA        : rv64imafdch
Boot HART ISA Extensions  : time,sstc
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 16
Boot HART MIDELEG         : 0x0000000000001666
Boot HART MEDELEG         : 0x0000000000f0b509
...mm_init done!
...proc_init done!
2022 Hello RISC-V
[S] Supervisor Mode Timer Interrupt
I
I[S] Supervisor Mode Timer Interrupt
H
IH[S] Supervisor Mode Timer Interrupt
H
IHH[S] Supervisor Mode Timer Interrupt
O
IHHO[S] Supervisor Mode Timer Interrupt
O
IHHOO[S] Supervisor Mode Timer Interrupt
L
IHHOOL[S] Supervisor Mode Timer Interrupt
L
IHHOOLL[S] Supervisor Mode Timer Interrupt
L
IHHOOLLL[S] Supervisor Mode Timer Interrupt
N
IHHOOLLLN[S] Supervisor Mode Timer Interrupt
N
IHHOOLLLNN[S] Supervisor Mode Timer Interrupt
N
IHHOOLLLNNN[S] Supervisor Mode Timer Interrupt
```

```
B
IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
B
IHHOOLLLNNNBB[S] Supervisor Mode Timer Interrupt
B
IHHOOLLLNNNBBB[S] Supervisor Mode Timer Interrupt
B
IHHOOLLLNNNBBBB[S] Supervisor Mode Timer Interrupt
M
IHHOOLLLNNNBBBBM[S] Supervisor Mode Timer Interrupt
M
IHHOOLLLNNNBBBBMM[S] Supervisor Mode Timer Interrupt
M
IHHOOLLLNNNBBBBMMM[S] Supervisor Mode Timer Interrupt
M
IHHOOLLLNNNBBBBMMMM[S] Supervisor Mode Timer Interrupt
E
IHHOOLLLNNNBBBBMMMME[S] Supervisor Mode Timer Interrupt
E
IHHOOLLLNNNBBBBMMMMEE[S] Supervisor Mode Timer Interrupt
E
IHHOOLLLNNNBBBBMMMMEEE[S] Supervisor Mode Timer Interrupt
E
IHHOOLLLNNNBBBBMMMMEEEE[S] Supervisor Mode Timer Interrupt
E
IHHOOLLLNNNBBBBMMMMEEEEE[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEP[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEPP[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEPPP[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEPPPP[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEPPPPP[S] Supervisor Mode Timer Interrupt
P
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPP[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJ[S] Supervisor Mode Timer Interrupt
J
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJ[S] Supervisor Mode Timer Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJD[S] Supervisor Mode Timer Interrupt
D
```

```
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDD[S] Supervisor Mode Timer Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDD[S] Supervisor Mode Timer Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDD[S] Supervisor Mode Timer
Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDD[S] Supervisor Mode Timer
Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDD[S] Supervisor Mode Timer
Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDD[S] Supervisor Mode Timer
Interrupt
D
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDD[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDC[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCC[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCC[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCC[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCC[S] Supervisor Mode Timer
Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCC[S] Supervisor Mode
Timer Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCC[S] Supervisor Mode
Timer Interrupt
C
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCC[S] Supervisor Mode
Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCG[S] Supervisor Mode
Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGG[S] Supervisor Mode
Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGG[S] Supervisor
Mode Timer Interrupt
G
```

```
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGG[S] Supervisor Mode Timer Interrupt
G
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGG[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKK[S] Supervisor Mode Timer Interrupt
K
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKK[S] Supervisor Mode Timer Interrupt
F
```

```
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
F[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
IHHOOLLLNNNBBBBMMMMEEEEEPPPPPPJJJJJJJJDDDDDDDDCCCCCCCCCGGGGGGGGGGKKKKKKKKKK
FFFFFFFFFFFF
NR_TASKS = 16, SJF test passed!
```

- 优先级调度算法输出测试结果：`#define NR_TASTS(1+15)`

```
Launch the qemu ......

OpenSBI v1.2
   ____                   _____ ____ _____
  / __ \                 / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name             : riscv-virtio,qemu
Platform Features         : medeleg
Platform HART Count       : 1
Platform IPI Device       : aclint-mswi
```

```
Platform Timer Device     : aclint-mtimer @ 10000000Hz
Platform Console Device   : uart8250
Platform HSM Device       : ---
Platform PMU Device       : ---
Platform Reboot Device    : sifive_test
Platform Shutdown Device  : sifive_test
Firmware Base             : 0x80000000
Firmware Size             : 212 KB
Runtime SBI Version       : 1.0

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*
Domain0 Region00          : 0x0000000002000000-0x000000000200ffff (I)
Domain0 Region01          : 0x0000000080000000-0x000000008003ffff ()
Domain0 Region02          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x0000000080200000
Domain0 Next Arg1         : 0x0000000087e00000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain          : root
Boot HART Priv Version    : v1.12
Boot HART Base ISA        : rv64imafdch
Boot HART ISA Extensions  : time,sstc
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 16
Boot HART MIDELEG         : 0x0000000000001666
Boot HART MEDELEG         : 0x0000000000f0b509
...mm_init done!
...proc_init done!
2022 Hello RISC-V
[S] Supervisor Mode Timer Interrupt
F
F[S] Supervisor Mode Timer Interrupt
F
FF[S] Supervisor Mode Timer Interrupt
F
FFF[S] Supervisor Mode Timer Interrupt
F
FFFF[S] Supervisor Mode Timer Interrupt
F
FFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
```

```
FFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFFFFFF[S] Supervisor Mode Timer Interrupt
F
FFFFFFFFFFF[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKKKKKK[S] Supervisor Mode Timer Interrupt
K
FFFFFFFFFFFKKKKKKKKKK[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGGGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGGGGGG[S] Supervisor Mode Timer Interrupt
G
FFFFFFFFFFFKKKKKKKKKKGGGGGGGGG[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFKKKKKKKKKKGGGGGGGGGC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFKKKKKKKKKKGGGGGGGGGCC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFKKKKKKKKKKGGGGGGGGGCCC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFKKKKKKKKKKGGGGGGGGGCCCC[S] Supervisor Mode Timer Interrupt
```

```
C
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCC[S] Supervisor Mode Timer Interrupt
C
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCC[S] Supervisor Mode Timer
Interrupt
C
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCC[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDD[S] Supervisor Mode Timer
Interrupt
D
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDD[S] Supervisor Mode Timer
Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJ[S] Supervisor Mode
Timer Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJJ[S] Supervisor Mode
Timer Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJJJ[S] Supervisor Mode
Timer Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJJJJ[S] Supervisor Mode
Timer Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJJJJJ[S] Supervisor Mode
Timer Interrupt
J
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGCCCCCCCCCDDDDDDDJJJJJJ[S] Supervisor Mode
Timer Interrupt
J
```

```
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJ[S] Supervisor
Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJP[S] Supervisor
Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPP[S] Supervisor
Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPP[S] Supervisor
Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPP[S] Supervisor
Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPP[S]
Supervisor Mode Timer Interrupt
P
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPP[S]
Supervisor Mode Timer Interrupt
E
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPE[S]
Supervisor Mode Timer Interrupt
E
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEE[S]
Supervisor Mode Timer Interrupt
E
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEE[S]
Supervisor Mode Timer Interrupt
E
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEE[S]
Supervisor Mode Timer Interrupt
E
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEE[S]
Supervisor Mode Timer Interrupt
M
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEM[S]
Supervisor Mode Timer Interrupt
M
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEMM[S]
Supervisor Mode Timer Interrupt
M
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEMMM[S]
Supervisor Mode Timer Interrupt
M
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEMMMM[S]
Supervisor Mode Timer Interrupt
B
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEMMMMB[S
] Supervisor Mode Timer Interrupt
B
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGCCCCCCCCCDDDDDDDDJJJJJJJPPPPPPEEEEEMMMMBB[
S] Supervisor Mode Timer Interrupt
B
```

```
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
[S] Supervisor Mode Timer Interrupt
B
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
B[S] Supervisor Mode Timer Interrupt
N
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BN[S] Supervisor Mode Timer Interrupt
N
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNN[S] Supervisor Mode Timer Interrupt
N
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNN[S] Supervisor Mode Timer Interrupt
L
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNL[S] Supervisor Mode Timer Interrupt
L
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLL[S] Supervisor Mode Timer Interrupt
L
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLL[S] Supervisor Mode Timer Interrupt
O
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLLO[S] Supervisor Mode Timer Interrupt
O
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLLOO[S] Supervisor Mode Timer Interrupt
H
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLLOOH[S] Supervisor Mode Timer Interrupt
H
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLLOOHH[S] Supervisor Mode Timer Interrupt
I
FFFFFFFFFFFFKKKKKKKKKKKKGGGGGGGGGGGGCCCCCCCCCCDDDDDDDDJJJJJJJJPPPPPPEEEEEMMMMBBB
BNNNLLLOOHHI
NR_TASKS = 16, PRIORITY test passed!
```

4 思考题

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？

Solution: 因为在本次实验中，线程切换是通过调用函数实现的，所有的`Caller Saved Register`将被保存到当前的栈上，实际上我们只是将PCB里的信息(`struct task_struct`)进行了save和restores。而在中断中，我们则需要保存当前执行环境的所有context.

在 `context_switch` 时，需要将就进程的相关信息即PCB入栈，且需要保存返回地址，因此在相关函数中不仅需要保存 `Callee Saved Register` ，还需要保存 `ra` 寄存器（返回地址）与 `sp` 寄存器（栈指针），共计保存 14 个寄存器。

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)。

Solution:

- 首先使用`disassemble __switch_to`查看函数`__switch_to`的地址，发现`ra`的保存与恢复在地址`0x0000000080200160`和`0x0000000080200198`处

```
0x000000008020018c <+44>:    sd      s9,136(a0)
0x0000000080200190 <+48>:    sd      s10,144(a0)
0x0000000080200194 <+52>:    sd      s11,152(a0)
0x0000000080200198 <+56>:    ld      ra,48(a1)
0x000000008020019c <+60>:    ld      sp,56(a1)
0x00000000802001a0 <+64>:    ld      s0,64(a1)
0x00000000802001a4 <+68>:    ld      s1,72(a1)
0x00000000802001a8 <+72>:    ld      s2,80(a1)
0x00000000802001ac <+76>:    ld      s3,88(a1)
0x00000000802001b0 <+80>:    ld      s4,96(a1)
0x00000000802001b4 <+84>:    ld      s5,104(a1)
0x00000000802001b8 <+88>:    ld      s6,112(a1)
0x00000000802001bc <+92>:    ld      s7,120(a1)
0x00000000802001c0 <+96>:    ld      s8,128(a1)
0x00000000802001c4 <+100>:   ld      s9,136(a1)
0x00000000802001c8 <+104>:   ld      s10,144(a1)
0x00000000802001cc <+108>:   ld      s11,152(a1)
0x00000000802001d0 <+112>:   ret
```

- 于是我们给这两个地址打上断点，然后进行测试
  - 在第一次切换时，可以观察到`$ra`寄存器的值在保存和恢复时的值分别是`switch_to+92`和`__dummy`

```
Breakpoint 1, __switch_to () at entry.S:100
100         sd ra, 48(a0)
(gdb) i r ra
ra             0x802008f8       0x802008f8 <switch_to+92>
(gdb) c
Continuing.

Breakpoint 2, __switch_to () at entry.S:115
115         ld ra, 48(a1)
(gdb) si
116         ld sp, 56(a1)
(gdb) i r ra
ra             0x802001d4       0x802001d4 <__dummy>
```

  - 在后续调度中，`$ra`寄存器的保存和恢复时的值均为`switch_to+92`

```
Breakpoint 1, __switch_to () at entry.S:100
100         sd ra, 48(a0)
(gdb) i r ra
ra             0x802008f8       0x802008f8 <switch_to+92>
(gdb) c
Continuing.

Breakpoint 2, __switch_to () at entry.S:115
115         ld ra, 48(a1)
(gdb) si
116         ld sp, 56(a1)
(gdb) i r ra
ra             0x802008f8       0x802008f8 <switch_to+92>
```

5 心得体会

在完成Lab 2: RV64 内核线程调度的过程中，整体而言，并不是十分顺利，主要来源于几个方面的Bug，值得反思。

- 不同的类型数之间的比较，在实现优先级调度的时候，对于某整型变量声明为-1，后在与`current->counter`进行比较的时候，出现比较问题
- 测试代码中，由于声明的`current`变量为`char []`类型，具体在进行测试时，发现强制类型转换在本机环境下出现问题
- 在Debug中，发现单步调试不会进入`dummy()`函数，导致该函数中的部分语句对`current->counter`进行的更改没有被检测到，在查询counter变化的过程中踩了不少坑

总的来说，由于较长时间的Debug，我对于内核线程的调度过程有了较为详尽的认识，也对gdb调试以及内核编程有了更深入的了解