

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

```
1 | .
2 | └─ arch
3 |   └─ riscv
4 |       └─ include
5 |           └─ mm.h
6 |           └─ proc.h
```

```

7 |         └─ kernel
8 |             └─ mm.c
9 |             └─ proc.c
10 | └─ include
11 |     └─ rand.h
12 |     └─ string.h
13 |     └─ test.h
14 |     └─ schedule_test.h
15 |
16 | └─ test
17 |     └─ schedule_null.c
18 |     └─ schedule_test.c
19 |     └─ Makefile
20 |
21 | └─ lib
22 |     └─ rand.c
23 |     └─ string.c
24 |
25 | └─ 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：

```

1 | #define PHY_START 0x0000000080000000
2 | #define PHY_SIZE 128 * 1024 * 1024 // 128MB, QEMU 默认内存大小
3 | #define PHY_END (PHY_START + PHY_SIZE)
4 |
5 | #define PGSIZE 0x1000 // 4KB
6 | #define PGROUNDUP(addr) ((addr + PGSIZE - 1) & ~(PGSIZE - 1))
7 | #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

```

1 | // arch/riscv/include/proc.h
2 |
3 | #include "types.h"
4 |
5 | #define NR_TASKS (1 + 31) // 用于控制 最大线程数量 (idle 线程 + 31 内核线程)
6 |
7 | #define TASK_RUNNING 0 // 为了简化实验，所有的线程都只有一种状态
8 |
9 | #define PRIORITY_MIN 1
10 | #define PRIORITY_MAX 10
11 |

```

```

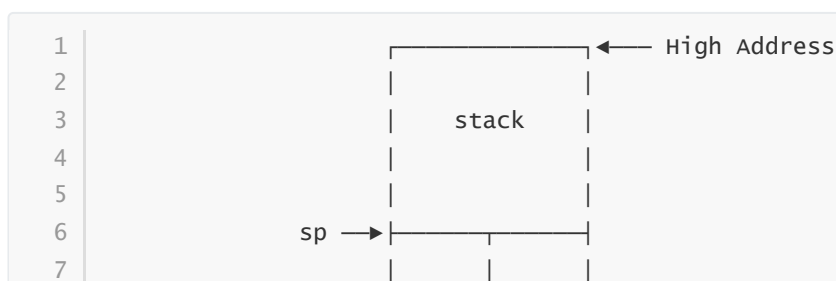
12  /* 用于记录`线程`的`内核栈与用户栈指针` */
13  /* (lab3中无需考虑, 在这里引入是为了之后实验的使用) */
14  struct thread_info {
15      uint64 kernel_sp;
16      uint64 user_sp;
17  };
18
19  /* 线程状态段数据结构 */
20  struct thread_struct {
21      uint64 ra;
22      uint64 sp;
23      uint64 s[12];
24  };
25
26  /* 线程数据结构 */
27  struct task_struct {
28      struct thread_info* thread_info;
29      uint64 state;    // 线程状态
30      uint64 counter;  // 运行剩余时间
31      uint64 priority; // 运行优先级 1最低 10最高
32      uint64 pid;      // 线程id
33
34      struct thread_struct thread;
35  };
36
37  /* 线程初始化 创建 NR_TASKS 个线程 */
38  void task_init();
39
40  /* 在时钟中断处理中被调用 用于判断是否需要调度 */
41  void do_timer();
42
43  /* 调度程序 选择出下一个运行的线程 */
44  void schedule();
45
46  /* 线程切换入口函数*/
47  void switch_to(struct task_struct* next);
48
49  /* 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 */
50  void dummy();

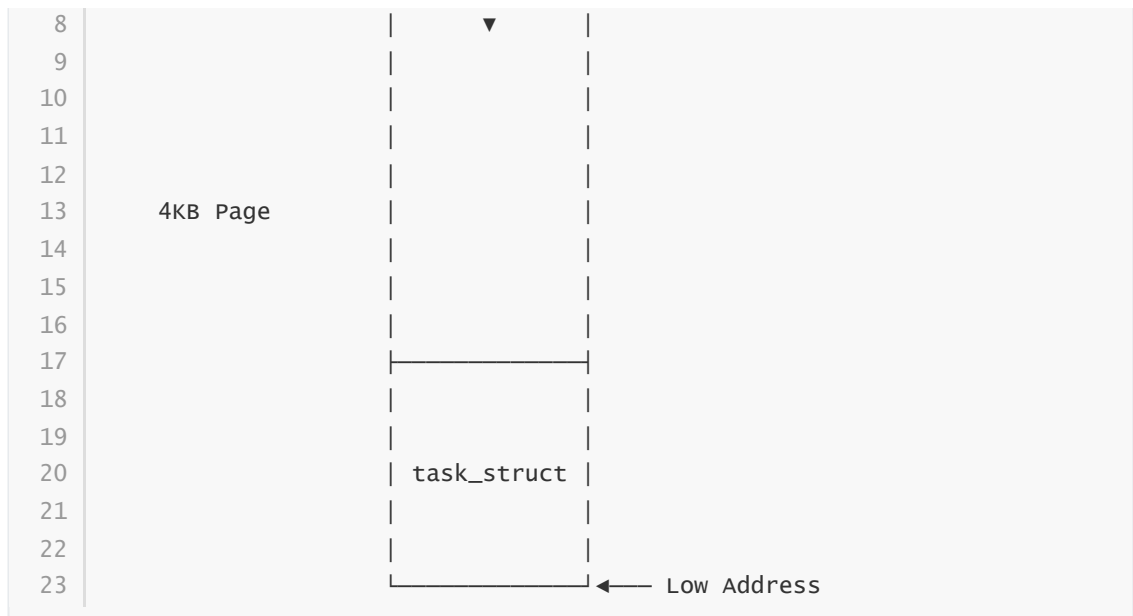
```

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:

```

1 void task_init() {
2     test_init(NR_TASKS);
3     // 1. 调用 kalloc() 为 idle 分配一个物理页
4     idle = (struct task_struct *)kalloc();
5     // 2. 设置 state 为 TASK_RUNNING;
6     idle->state = TASK_RUNNING;
7     // 3. 由于 idle 不参与调度 可以将其 counter / priority 设置为 0
8     idle->counter = 0;
9     idle->priority = 0;
10    // 4. 设置 idle 的 pid 为 0
11    idle->pid = 0;
12    // 5. 将 current 和 task[0] 指向 idle
13    task[0] = idle;
14    current = idle;
15    /* YOUR CODE HERE */
16
17    // 1. 参考 idle 的设置, 为 task[1] ~ task[NR_TASKS - 1] 进行初始化
18    for(int i = 1; i < NR_TASKS; i++){
19        task[i] = (struct task_struct *)kalloc();
20        task[i]->state = TASK_RUNNING;
21        task[i]->pid = i;
22        task[i]->counter = task_test_counter[i];
23        task[i]->priority = task_test_priority[i];
24        // printk("task[i]->pid: %d\n", task[i]->pid);
25        // printk("task[i]->counter: %d\n", task[i]->counter);
26        // printk("task[i]->priority: %d\n", task[i]->priority);
27
28        task[i]->thread.ra = (uint64)__dummy;

```

```

29     task[i]->thread.sp = (uint64)task[i] + 4096;
30
31     }
32     // 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:
33     //     task[i].counter = task_test_counter[i];
34     //     task[i].priority = task_test_priority[i];
35     // 3. 为 task[1] ~ task[NR_TASKS - 1] 设置 `thread_struct` 中的 `ra` 和
`sp`,
36     // 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
37
38     printk("...proc_init done!\n");
39 }

```

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() :

```

1 // arch/riscv/kernel/proc.c
2
3 void dummy() {
4     schedule_test();
5     uint64 MOD = 1000000007;
6     uint64 auto_inc_local_var = 0;
7     int last_counter = -1;
8     while(1) {
9         if ((last_counter == -1 || current->counter != last_counter) &&
current->counter > 0) {
10             if(current->counter == 1){
11                 --(current->counter); // forced the counter to be
zero if this thread is going to be scheduled
12             } // in case that the new counter
is also 1, leading the information not printed.
13             last_counter = current->counter;
14             auto_inc_local_var = (auto_inc_local_var + 1) % MOD;
15             printk("[PID = %d] is running. auto_inc_local_var = %d\n",
current->pid, auto_inc_local_var);
16         }
17     }
18 }

```

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

```
1 # arch/riscv/kernel/entry.S
2
3     .global __dummy
4 __dummy:
5     la t0, dummy
6     csrw sepc, t0
7     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

```
1 // arch/riscv/kernel/proc.c
2
3 extern void __switch_to(struct task_struct* prev, struct task_struct*
next);
4
5
6 void switch_to(struct task_struct* next) {
7     if(next->pid == current->pid)
8         return;
9     // printk("current_pid: %d\n", current->pid);
10    // printk("Next_pid: %d\n", next->pid);
11
12    //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
13    struct task_struct* prev = current;
14    current = next;
15    // 调用 __switch_to 进行线程切换
16    __switch_to(prev, next);
17 }
```

- 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

```
1 # arch/riscv/kernel/entry.S
2
3     .globl __switch_to
4 __switch_to:
5     # task_struct = thread_info + state + counter + priority + pid +
thread_info
6     # thread initial address: task_struct + 4 * 8 + 2 * 8 = prev + 48
7     sd ra, 48(a0)
```

```

8      sd sp, 56(a0)
9      sd s0, 64(a0)
10     sd s1, 72(a0)
11     sd s2, 80(a0)
12     sd s3, 88(a0)
13     sd s4, 96(a0)
14     sd s5, 104(a0)
15     sd s6, 112(a0)
16     sd s7, 120(a0)
17     sd s8, 128(a0)
18     sd s9, 136(a0)
19     sd s10, 144(a0)
20     sd s11, 152(a0)
21
22     ld ra, 48(a1)
23     ld sp, 56(a1)
24     ld s0, 64(a1)
25     ld s1, 72(a1)
26     ld s2, 80(a1)
27     ld s3, 88(a1)
28     ld s4, 96(a1)
29     ld s5, 104(a1)
30     ld s6, 112(a1)
31     ld s7, 120(a1)
32     ld s8, 128(a1)
33     ld s9, 136(a1)
34     ld s10, 144(a1)
35     ld s11, 152(a1)
36
37     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

```

1  void do_timer() {
2      if(current->pid == 0)
3          schedule();
4      else{
5          // printk("current_pid: %d\n", current->pid);
6          // printk("current_counter: %d\n", current->counter);
7          current->counter--;
8          if(current->counter == 0)
9              schedule();
10         else{
11             return;
12         }
13     }
14 }

```

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：

```
1 // arch/riscv/kernel/proc.c
2
3 void schedule(void) {
4     int min_i = 0, cnt = 0;
5     while(1){
6         int min_counter = 1000;
7         min_i = 1, cnt = 0;
8         for(int i = 1; i < NR_TASKS; i++)
9         {
10             if(task[i]->counter == 0)
11                 cnt++;
12             if(task[i]->state == TASK_RUNNING && task[i]->counter > 0 &&
task[i]->counter < min_counter)
13             {
14                 min_counter = task[i]->counter;
15                 min_i = i;
16             }
17         }
18         if(cnt == NR_TASKS - 1)
19         {
20             for(int i = 1; i < NR_TASKS; i++)
21                 task[i]->counter = rand() % 10;
22         }
23         else
24             break;
25     }
26     // printk("schedule_next_pid: %d\n", task[min_i]->pid);
27     if(min_i != 0)
28         switch_to(task[min_i]);
29     else
30         printk("Error occured when schedule!\n");
31 }
```

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：


```

1 // arch/riscv/kernel/proc.c
2
3 void schedule(void) {
4     int c, next, i;
5     while(1){
6         // printk("why you are stunned here\n");
7         c = 0;
8         next = 0;
9         i = NR_TASKS;
10        struct task_struct ** p = &task[NR_TASKS];
11        while(--i > 0){
12            // printk("pid: %d\n", task[i]->pid);
13            // printk("state: %d\n", task[i]->state);
14            // printk("counter: %d\n", task[i]->counter);
15            if( task[i]->state == TASK_RUNNING && task[i]->counter > c)
16            {
17                c = (task[i])->counter;
18                next = i;
19            }
20        }
21        // printk("c: %d\n", c);
22        if(c > 0) break;
23        for(int j = 1; j <= NR_TASKS; j++)
24            if (task[j])
25                task[j]->counter = (task[j]->counter >> 1) + task[j]-
>priority;
26    }
27    // printk("next :%d\n", next);
28    if(next != current->pid && next != 0)
29        switch_to(task[next]);
30 }

```

4.4 编译及测试

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

```

1 Launch the qemu .....
2
3 OpenSBI v1.2
4
5      _____
6      /  _  \                /  _  |  _  \  _  |
7      |  |  |  _  _  _  _  | (  _  |  _  |  |  |
8      |  |  |  '  \  /  _  \  '  \  \  _  \  |  _  <  |  |
9      |  |  |  |  _  |  _  /  |  |  |  _  _  |  |  _  |  |  _  |
10     \  _  /  |  _  /  \  _  |  |  |  _  _  /  |  _  /  _  _  |
11         |  |
12         |  |
13 Platform Name           : riscv-virtio,qemu
14 Platform Features      : medeleg
15 Platform HART Count    : 1
16 Platform IPI Device    : aclint-mswi
17 Platform Timer Device   : aclint-mtimer @ 10000000Hz
18 Platform Console Device : uart8250
19 Platform HSM Device    : ---

```

```
20 Platform PMU Device      : ---
21 Platform Reboot Device   : sifive_test
22 Platform Shutdown Device : sifive_test
23 Firmware Base            : 0x80000000
24 Firmware Size            : 212 KB
25 Runtime SBI Version      : 1.0
26
27 Domain0 Name             : root
28 Domain0 Boot HART        : 0
29 Domain0 HARTs            : 0*
30 Domain0 Region00         : 0x0000000002000000-0x000000000200ffff (I)
31 Domain0 Region01         : 0x0000000008000000-0x0000000008003ffff (C)
32 Domain0 Region02         : 0x0000000000000000-0xffffffffffffffff (R,W,X)
33 Domain0 Next Address     : 0x0000000008020000
34 Domain0 Next Arg1        : 0x00000000087e0000
35 Domain0 Next Mode        : S-mode
36 Domain0 SysReset         : yes
37
38 Boot HART ID             : 0
39 Boot HART Domain         : root
40 Boot HART Priv Version   : v1.12
41 Boot HART Base ISA       : rv64imafdc
42 Boot HART ISA Extensions : time,sstc
43 Boot HART PMP Count      : 16
44 Boot HART PMP Granularity : 4
45 Boot HART PMP Address Bits : 54
46 Boot HART MHPM Count     : 16
47 Boot HART MIDELEG        : 0x0000000000001666
48 Boot HART MEDELEG        : 0x0000000000f0b509
49 ...mm_init done!
50 ...proc_init done!
51 2022 Hello RISC-V
52 [S] Supervisor Mode Timer Interrupt
53 I
54 I[S] Supervisor Mode Timer Interrupt
55 H
56 IH[S] Supervisor Mode Timer Interrupt
57 H
58 IHH[S] Supervisor Mode Timer Interrupt
59 O
60 IHHO[S] Supervisor Mode Timer Interrupt
61 O
62 IHHOO[S] Supervisor Mode Timer Interrupt
63 L
64 IHHOOL[S] Supervisor Mode Timer Interrupt
65 L
66 IHHOOLL[S] Supervisor Mode Timer Interrupt
67 L
68 IHHOOLLL[S] Supervisor Mode Timer Interrupt
69 N
70 IHHOOLLLN[S] Supervisor Mode Timer Interrupt
71 N
72 IHHOOLLLNN[S] Supervisor Mode Timer Interrupt
73 N
74 IHHOOLLLNNN[S] Supervisor Mode Timer Interrupt
```

75	B
76	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
77	B
78	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
79	B
80	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
81	B
82	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
83	M
84	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
85	M
86	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
87	M
88	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
89	M
90	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
91	E
92	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
93	E
94	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
95	E
96	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
97	E
98	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
99	E
100	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
101	P
102	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
103	P
104	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
105	P
106	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
107	P
108	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
109	P
110	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
111	P
112	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
113	J
114	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
115	J
116	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
117	J
118	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
119	J
120	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
121	J
122	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
123	J
124	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
125	J
126	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
127	D
128	IHHOOLLLNNNB[S] Supervisor Mode Timer Interrupt
129	D

130 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDD[S] Supervisor Mode Timer Interrupt
131 D
132 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDD[S] Supervisor Mode Timer Interrupt
133 D
134 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDD[S] Supervisor Mode Timer
Interrupt
135 D
136 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDD[S] Supervisor Mode Timer
Interrupt
137 D
138 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDD[S] Supervisor Mode Timer
Interrupt
139 D
140 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDD[S] Supervisor Mode Timer
Interrupt
141 D
142 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDD[S] Supervisor Mode Timer
Interrupt
143 C
144 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDC[S] Supervisor Mode Timer
Interrupt
145 C
146 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCC[S] Supervisor Mode Timer
Interrupt
147 C
148 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCC[S] Supervisor Mode Timer
Interrupt
149 C
150 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCC[S] Supervisor Mode Timer
Interrupt
151 C
152 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCC[S] Supervisor Mode Timer
Interrupt
153 C
154 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCC[S] Supervisor Mode
Timer Interrupt
155 C
156 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCC[S] Supervisor Mode
Timer Interrupt
157 C
158 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCC[S] Supervisor Mode
Timer Interrupt
159 C
160 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCC[S] Supervisor Mode
Timer Interrupt
161 G
162 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCG[S] Supervisor Mode
Timer Interrupt
163 G
164 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGG[S] Supervisor Mode
Timer Interrupt
165 G
166 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGG[S] Supervisor
Mode Timer Interrupt
167 G

168 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGG[S] Supervisor
Mode Timer Interrupt
169 G
170 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGG[S] Supervisor
Mode Timer Interrupt
171 G
172 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGG[S] Supervisor
Mode Timer Interrupt
173 G
174 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGG[S] Supervisor
Mode Timer Interrupt
175 G
176 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGG[S]
Supervisor Mode Timer Interrupt
177 G
178 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGG[S]
Supervisor Mode Timer Interrupt
179 G
180 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGG[S]
Supervisor Mode Timer Interrupt
181 K
182 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGK[S]
Supervisor Mode Timer Interrupt
183 K
184 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKK[S]
Supervisor Mode Timer Interrupt
185 K
186 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKK[S]
Supervisor Mode Timer Interrupt
187 K
188 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKK[S]
Supervisor Mode Timer Interrupt
189 K
190 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKK[S]
Supervisor Mode Timer Interrupt
191 K
192 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKK[S]
Supervisor Mode Timer Interrupt
193 K
194 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKKK[S]
Supervisor Mode Timer Interrupt
195 K
196 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKKKK[S]
Supervisor Mode Timer Interrupt
197 K
198 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKKKKK[S]
] Supervisor Mode Timer Interrupt
199 K
200 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKKKKKK[S]
S] Supervisor Mode Timer Interrupt
201 K
202 IHHOULLNNBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGGKKKKKKKKKKK
[S] Supervisor Mode Timer Interrupt
203 F

```

204 IHHOULLNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    F[S] Supervisor Mode Timer Interrupt
205 F
206 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FF[S] Supervisor Mode Timer Interrupt
207 F
208 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFF[S] Supervisor Mode Timer Interrupt
209 F
210 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFF[S] Supervisor Mode Timer Interrupt
211 F
212 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFF[S] Supervisor Mode Timer Interrupt
213 F
214 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFF[S] Supervisor Mode Timer Interrupt
215 F
216 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFF[S] Supervisor Mode Timer Interrupt
217 F
218 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFFF[S] Supervisor Mode Timer Interrupt
219 F
220 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFFF[S] Supervisor Mode Timer Interrupt
221 F
222 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFFF[S] Supervisor Mode Timer Interrupt
223 F
224 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFFF[S] Supervisor Mode Timer Interrupt
225 F
226 IHHOULLNNNNBBBBMMMMEEEEPPPPPPJJJJJJDDDDDDDDCCCCCCCCGGGGGGGGGKKKKKKKKKK
    FFFFFFFF[S] Supervisor Mode Timer Interrupt
227 NR_TASKS = 16, SJF test passed!

```

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

```

1 Launch the qemu .....
2
3 OpenSBI v1.2
4
5  _ _ _ _ _
6 / _ \      / _ \ | _ \ _ \
7 | | | | _ _ _ _ _ | ( _ | | ) | |
8 | | | | ' _ \ / _ \ ' _ \ \ _ \ | _ < | |
9 | | | | | _ \ | _ \ / _ \ | _ \ | _ \
10  \ _ \ / | _ \ \ _ \ | _ \ / _ \ / _ \
11      | |
12      | _ |
13 Platform Name           : riscv-virtio,qemu
14 Platform Features       : medeleg
15 Platform HART Count     : 1
16 Platform IPI Device     : aclint-mswi

```

```
17 Platform Timer Device      : aclint-mtimer @ 100000000Hz
18 Platform Console Device    : uart8250
19 Platform HSM Device        : ---
20 Platform PMU Device        : ---
21 Platform Reboot Device     : sifive_test
22 Platform Shutdown Device   : sifive_test
23 Firmware Base              : 0x80000000
24 Firmware Size              : 212 KB
25 Runtime SBI Version        : 1.0
26
27 Domain0 Name               : root
28 Domain0 Boot HART          : 0
29 Domain0 HARTs              : 0*
30 Domain0 Region00           : 0x0000000002000000-0x000000000200ffff (I)
31 Domain0 Region01           : 0x0000000008000000-0x0000000008003ffff ( )
32 Domain0 Region02           : 0x0000000000000000-0xffffffffffffffff (R,W,X)
33 Domain0 Next Address       : 0x0000000080200000
34 Domain0 Next Arg1          : 0x0000000087e00000
35 Domain0 Next Mode          : S-mode
36 Domain0 SysReset           : yes
37
38 Boot HART ID               : 0
39 Boot HART Domain           : root
40 Boot HART Priv Version     : v1.12
41 Boot HART Base ISA         : rv64imafdc
42 Boot HART ISA Extensions   : time,sstc
43 Boot HART PMP Count        : 16
44 Boot HART PMP Granularity  : 4
45 Boot HART PMP Address Bits : 54
46 Boot HART MHPM Count       : 16
47 Boot HART MIDELEG          : 0x0000000000001666
48 Boot HART MEDELEG          : 0x0000000000f0b509
49 ...mm_init done!
50 ...proc_init done!
51 2022 Hello RISC-V
52 [S] Supervisor Mode Timer Interrupt
53 F
54 F[S] Supervisor Mode Timer Interrupt
55 F
56 FF[S] Supervisor Mode Timer Interrupt
57 F
58 FFF[S] Supervisor Mode Timer Interrupt
59 F
60 FFFF[S] Supervisor Mode Timer Interrupt
61 F
62 FFFFF[S] Supervisor Mode Timer Interrupt
63 F
64 FFFFFF[S] Supervisor Mode Timer Interrupt
65 F
66 FFFFFFF[S] Supervisor Mode Timer Interrupt
67 F
68 FFFFFFFF[S] Supervisor Mode Timer Interrupt
69 F
70 FFFFFFFF[S] Supervisor Mode Timer Interrupt
71 F
```

```
72 FFFFFFFFFF[S] Supervisor Mode Timer Interrupt
73 F
74 FFFFFFFFFF[S] Supervisor Mode Timer Interrupt
75 F
76 FFFFFFFFFF[S] Supervisor Mode Timer Interrupt
77 K
78 FFFFFFFFFFK[S] Supervisor Mode Timer Interrupt
79 K
80 FFFFFFFFFFKK[S] Supervisor Mode Timer Interrupt
81 K
82 FFFFFFFFFFKKK[S] Supervisor Mode Timer Interrupt
83 K
84 FFFFFFFFFFKKKK[S] Supervisor Mode Timer Interrupt
85 K
86 FFFFFFFFFFKKKK[S] Supervisor Mode Timer Interrupt
87 K
88 FFFFFFFFFFKKKKK[S] Supervisor Mode Timer Interrupt
89 K
90 FFFFFFFFFFKKKKKK[S] Supervisor Mode Timer Interrupt
91 K
92 FFFFFFFFFFKKKKKKK[S] Supervisor Mode Timer Interrupt
93 K
94 FFFFFFFFFFKKKKKKKK[S] Supervisor Mode Timer Interrupt
95 K
96 FFFFFFFFFFKKKKKKKKK[S] Supervisor Mode Timer Interrupt
97 K
98 FFFFFFFFFFKKKKKKKKKK[S] Supervisor Mode Timer Interrupt
99 G
100 FFFFFFFFFFKKKKKKKKKKG[S] Supervisor Mode Timer Interrupt
101 G
102 FFFFFFFFFFKKKKKKKKKKGG[S] Supervisor Mode Timer Interrupt
103 G
104 FFFFFFFFFFKKKKKKKKKKGGG[S] Supervisor Mode Timer Interrupt
105 G
106 FFFFFFFFFFKKKKKKKKKKGGGG[S] Supervisor Mode Timer Interrupt
107 G
108 FFFFFFFFFFKKKKKKKKKKGGGGG[S] Supervisor Mode Timer Interrupt
109 G
110 FFFFFFFFFFKKKKKKKKKKGGGGGG[S] Supervisor Mode Timer Interrupt
111 G
112 FFFFFFFFFFKKKKKKKKKKGGGGGGG[S] Supervisor Mode Timer Interrupt
113 G
114 FFFFFFFFFFKKKKKKKKKKGGGGGGGG[S] Supervisor Mode Timer Interrupt
115 G
116 FFFFFFFFFFKKKKKKKKKKGGGGGGGGG[S] Supervisor Mode Timer Interrupt
117 G
118 FFFFFFFFFFKKKKKKKKKKGGGGGGGGG[S] Supervisor Mode Timer Interrupt
119 C
120 FFFFFFFFFFKKKKKKKKKKGGGGGGGGGC[S] Supervisor Mode Timer Interrupt
121 C
122 FFFFFFFFFFKKKKKKKKKKGGGGGGGGGCC[S] Supervisor Mode Timer Interrupt
123 C
124 FFFFFFFFFFKKKKKKKKKKGGGGGGGGGCC[S] Supervisor Mode Timer Interrupt
125 C
126 FFFFFFFFFFKKKKKKKKKKGGGGGGGGGCC[S] Supervisor Mode Timer Interrupt
```



```
127 C
128 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCC[S] Supervisor Mode Timer Interrupt
129 C
130 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCC[S] Supervisor Mode Timer Interrupt
131 C
132 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCC[S] Supervisor Mode Timer Interrupt
133 C
134 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCC[S] Supervisor Mode Timer
Interrupt
135 C
136 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCC[S] Supervisor Mode Timer
Interrupt
137 D
138 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCD[S] Supervisor Mode Timer
Interrupt
139 D
140 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDD[S] Supervisor Mode Timer
Interrupt
141 D
142 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDD[S] Supervisor Mode Timer
Interrupt
143 D
144 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDD[S] Supervisor Mode Timer
Interrupt
145 D
146 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDD[S] Supervisor Mode Timer
Interrupt
147 D
148 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDD[S] Supervisor Mode Timer
Interrupt
149 D
150 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDD[S] Supervisor Mode Timer
Interrupt
151 D
152 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDD[S] Supervisor Mode Timer
Interrupt
153 J
154 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJ[S] Supervisor Mode
Timer Interrupt
155 J
156 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJJ[S] Supervisor Mode
Timer Interrupt
157 J
158 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJJJ[S] Supervisor Mode
Timer Interrupt
159 J
160 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJ[S] Supervisor Mode
Timer Interrupt
161 J
162 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJ[S] Supervisor Mode
Timer Interrupt
163 J
164 FFFFFFFFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJ[S] Supervisor Mode
Timer Interrupt
165 J
```

```

166 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJ[S] Supervisor
    Mode Timer Interrupt
167 P
168 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJJP[S] Supervisor
    Mode Timer Interrupt
169 P
170 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPP[S] Supervisor
    Mode Timer Interrupt
171 P
172 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPP[S] Supervisor
    Mode Timer Interrupt
173 P
174 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPP[S] Supervisor
    Mode Timer Interrupt
175 P
176 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPP[S]
    Supervisor Mode Timer Interrupt
177 P
178 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPP[S]
    Supervisor Mode Timer Interrupt
179 E
180 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPE[S]
    Supervisor Mode Timer Interrupt
181 E
182 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEE[S]
    Supervisor Mode Timer Interrupt
183 E
184 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEE[S]
    Supervisor Mode Timer Interrupt
185 E
186 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEE[S]
    Supervisor Mode Timer Interrupt
187 E
188 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEE[S]
    Supervisor Mode Timer Interrupt
189 M
190 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEM[S]
    Supervisor Mode Timer Interrupt
191 M
192 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEMM[S]
    Supervisor Mode Timer Interrupt
193 M
194 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEMMM[S]
    Supervisor Mode Timer Interrupt
195 M
196 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEMMMM[S]
    Supervisor Mode Timer Interrupt
197 B
198 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEMMMMB[S]
    ] Supervisor Mode Timer Interrupt
199 B
200 FFFFFFFFFFXXXXXXXXXXXXGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPPPEEEEEMMMMBB[
    S] Supervisor Mode Timer Interrupt
201 B

```

```

202 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
[S] Supervisor Mode Timer Interrupt
203 B
204 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
B[S] Supervisor Mode Timer Interrupt
205 N
206 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BN[S] Supervisor Mode Timer Interrupt
207 N
208 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNN[S] Supervisor Mode Timer Interrupt
209 N
210 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNN[S] Supervisor Mode Timer Interrupt
211 L
212 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNNL[S] Supervisor Mode Timer Interrupt
213 L
214 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLL[S] Supervisor Mode Timer Interrupt
215 L
216 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLL[S] Supervisor Mode Timer Interrupt
217 O
218 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLO[S] Supervisor Mode Timer Interrupt
219 O
220 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLOO[S] Supervisor Mode Timer Interrupt
221 H
222 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLOOH[S] Supervisor Mode Timer Interrupt
223 H
224 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLOOHH[S] Supervisor Mode Timer Interrupt
225 I
226 FFFFFFFFFFBBBBBBBBBBBBGGGGGGGGGGCCCCCCCCDDDDDDDDJJJJJJPPPPPEEEEEEMMMBBB
BNNLLOOHHI
227 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