

Assignment1 Report

12212320 谭政

Checkpoint. 1 (10 mins)

Checkpoint Passed. 你应该会看到日志中 8 个工作线程能轮流运行计数。但是，在出现第一个退出的进程

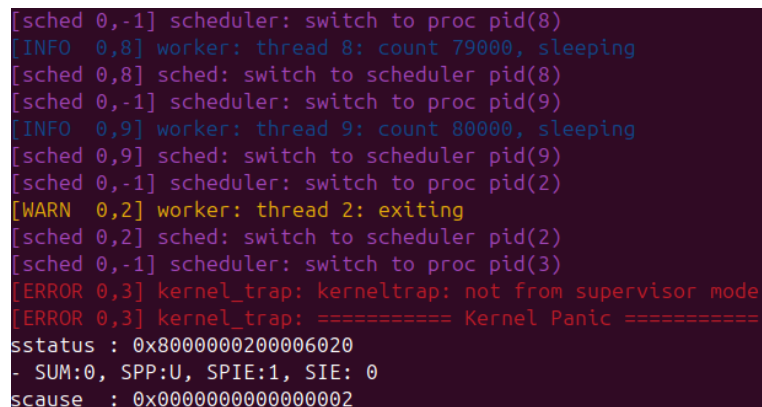
时，发生了 kernel panic。此时你应该遇到 kerneltrap: not from supervisor mode 的错误，以及 kernel panic。

这表示你已经完成了 Checkpoint. 1.

代码修改如下：

```
switch (exception_code) {
    case SupervisorTimer:
        tracef("s-timer interrupt, cycle: %d", r_time());
        set_next_timer();
        // we never preempt kernel threads.
        // Check if there is a running process
        if (curr_proc() != NULL) { (判断是否有进程在 CPU 上)
            // Temporarily clear mycpu()->inkernel_trap to avoid panic
            mycpu()->inkernel_trap--; (清零与重置)
            yield();
            mycpu()->inkernel_trap++;
        }
}
```

运行截图：



```
[sched 0,-1] scheduler: switch to proc pid(8)
[INFO 0,8] worker: thread 8: count 79000, sleeping
[sched 0,8] sched: switch to scheduler pid(8)
[sched 0,-1] scheduler: switch to proc pid(9)
[INFO 0,9] worker: thread 9: count 80000, sleeping
[sched 0,9] sched: switch to scheduler pid(9)
[sched 0,-1] scheduler: switch to proc pid(2)
[WARN 0,2] worker: thread 2: exiting
[sched 0,2] sched: switch to scheduler pid(2)
[sched 0,-1] scheduler: switch to proc pid(3)
[ERROR 0,3] kernel_trap: kerneltrap: not from supervisor mode
[ERROR 0,3] kernel_trap: ===== Kernel Panic =====
sstatus : 0x8000000200006020
- SUM:0, SPP:U, SPIE:1, SIE: 0
scause : 0x0000000000000002
```

```
00000000000000010
t4: 0x0000000080046dda   t5: 0x000000000000000f   t6: 0x0000000000000027
[PANIC 0,3] os/trap.c:92: kernel panic
```

Checkpoint. 2 (50mins)

观察 Kernel Panic 报告中的 `sstatus` 寄存器值, `SPP:U` 表示 `sstatus.SPP` 位是 0, 表示进入该 Trap 前, CPU 处于 U-mode。但是, 我们从未有代码预期会返回到 U-mode 执行; 并且, 返回到用户空间的唯一途径是在 `sret` 前清空 `sstatus` 的 `SPP` 位, 但是我们的代码从未有主动清空过该标志位。找到内核为什么会降级到 U-mode, 并解决该问题。

```
sstatus : 0x8000000200006020
- SUM:0, SPP:U, SPIE:1, SIE: 0
```

表示陷阱 (trap) 是从用户模式 (user mode) 触发的, 当执行 `SRET` 指令返回陷阱处理程序时, 如果 `SPP` 为 0, 则返回到用户模式; 如果 `SPP` 为 1, 则返回到内核模式。在返回时, `SPP` 会被清零。

根据提示可知, 经过了 `scheduler` 函数后在返回时, `SPP` 会被清零。于是在 `exit` 函数使用 `sched()`; 的前后, 使用 `w_sstatus(r_sstatus() | SSTATUS_SPP)`; 保证 `SPP` 为 1。

Code: 见 Checkpoint3

运行截图:

```
[INFO 0,1] init: init ends!
[PANIC 0,1] os/proc.c:225: init process exited
```

Checkpoint. 3 (3hours)

该 Checkpoint 需要你通过多 CPU 下的压力测试。

将 `sched.c` 中两条 `logf` 注释以避免大量输出; 将 `timer.h` 中对 `TICKS_PER_SEC` 改为 400, 提高中断的频率;

将 `nommu_init.c` 中的 `NTHREAD` 改为 15, 将 `SLEEP_TIME` 改为 50。

反复使用 `make runsmp` 启动多 CPU 的操作系统，你应该会遇到程序卡死或者 `kernel panic` 的问题。尝试解决该问题。

通过阅读文档可知 `sepc` 寄存器的值也应该不改变：在 `sched()` 中对 `sstatus` 和 `sepc` 寄存器的值进行保存和恢复。（把 `checkpoint2` 的实现也放一起）

Code:

```
void sched() {
    int interrupt_on;

    struct proc *p = curr_proc();
    uint64 sstatus = r_sstatus();
    uint64 sepc = r_sepc(); // Save sepc

    if (!holding(&p->lock))
        panic("not holding p->lock");
    if (mycpu()->noff != 1)
        panic("holding another locks");
    if (p->state == RUNNING)
        panic("sched running process");
    if (mycpu()->inkernel_trap)
        panic("sched should never be called in kernel trap context.");
    assert(!intr_get());

    interrupt_on = mycpu()->interrupt_on;
    // Save the current sstatus and sepc values
    uint64 saved_sstatus = r_sstatus();
    uint64 saved_sepc = r_sepc();
    // logf(PURPLE, "sched", "switch to scheduler pid(%d)", p->pid);
    swtch(&p->context, &mycpu()->sched_context);
    // Restore the saved sstatus and sepc values
    w_sstatus(saved_sstatus);
    w_sepc(saved_sepc);
    mycpu()->interrupt_on = interrupt_on;
    // if scheduler returns here: p->lock must be holding.
    if (!holding(&p->lock))
        panic("not holding p->lock after sched.swtch returns");
    w_sstatus(sstatus | SSTATUS_SPP); // Ensure SPP is 1
}
```

实验结果:

```
[INFO 0,1] init: thread 3 exited with code 23, expected 23
[INFO 0,1] init: thread 4 exited with code 24, expected 24
[INFO 0,1] init: thread 5 exited with code 25, expected 25
[WARN 3,13] worker: thread 13: exiting
[INFO 0,1] init: thread 6 exited with code 26, expected 26
[INFO 0,1] init: thread 7 exited with code 27, expected 27
[INFO 0,1] init: thread 8 exited with code 28, expected 28
[WARN 2,9] worker: thread 9: exiting
[INFO 2,1] init: thread 9 exited with code 29, expected 29
[INFO 2,1] init: thread 10 exited with code 30, expected 30
[INFO 2,1] init: thread 11 exited with code 31, expected 31
[INFO 2,1] init: thread 12 exited with code 32, expected 32
[INFO 2,1] init: thread 13 exited with code 33, expected 33
[INFO 2,1] init: thread 14 exited with code 34, expected 34
[INFO 2,1] init: thread 15 exited with code 35, expected 35
[INFO 2,1] init: thread 16 exited with code 36, expected 36
[INFO 2,1] init: all threads exited, count 150000

[INFO 2,1] init: init ends!
[PANIC 2,1] os/proc.c:225: init process exited
[PANIC 1,-1] os/trap.c:45: other CPU has panicked
[PANIC 3,-1] os/trap.c:45: other CPU has panicked
[PANIC 0,-1] os/trap.c:45: other CPU has panicked
```

实验总结：

完成时间：

本次实验共花费 4 小时。

Checkpoint 1+Checkpoint 2: 1 小时。

Checkpoint 3: 3 小时。

主要收获：

理解了 Trap 处理和上下文切换机制。

了解了多 CPU 环境下的竞争条件和锁机制。