Project3 Preemptive Kernel 设计文档

中国科学院大学
付琳晴
11.8.2017

1. 时钟中断与 blocking sleep 设计流程

1.1 中断处理流程

一般的流程是：STATUS 寄存器 IE 位（末位）清零关中断（STI），通过 CAUSE 寄存器判断是否是时钟中断，进入相应的中断处理函数，保存 CP0 寄存器与通用寄存器，保留现场，进行中断处理，将栈中存的寄存器的数据恢复，STATUS 寄存器末位置 1，开中断（CTI）。

在本次实验中，我们只需要处理时钟中断，因此如果其他中断进入，将中断位清 0 返回即可，不用处理。我在 handle_int 函数中实现的处理流程是判断是否是时钟中断，如果不是时钟中断，将 cause 寄存器中断位清 0 并返回。若是时钟中断则进入，并将 time_elapsed 加 1，因为时间中断是以固定周期发生的，就可以当做时间计数的标准，time_elapsed 我们需要在后续对睡眠进程做处理时用到。接着要清 COUNT 寄存器并对 COMPARE 寄存器重新赋值。之所以要这样做是因为在体系结构实验课时了解到，在 MIPS 的 CPU 中，时间中断的发生就对 COMPARE 寄存器附一个固定值，COUNT 寄存器会自动一个周期加两次，当两个寄存器值相等时会将 CAUSE.TI 位置 1，硬件将自动将 CAUSE.IP7 位置 1，即发生时间中断。因此我们重新对这两个寄存器赋值后，硬件会自动清除 CAUSE.TI 位（其实也可以直接清 CAUSE.IP7 位，只不过因为体系结构实验课对重写操作印象很深，就采用了这个方法）。接下来要调用 TEST_NESTED_COUNT 区分线程与进程，若是内核线程则退出中断，是进程则处理中断。我们这里处理中断是将当前任务放回任务队列，调度新的任务。调度之后恢复新任务的寄存器，中断处理即结束。

1.2 Blocking sleep

Blocking sleep 即将任务睡眠一段时间，当过了睡眠时间后再从睡眠队列中拿出放回 ready 队列。当 task 调用 sleep 函数时，会传入一个睡眠长度的参数，用这个参数加上当前时间就是该任务需要被唤醒的时候。用 check_sleeping 函数将 sleep 队列中需要被唤醒的任务都唤醒然后放回 ready 队列中。每一次用 scheduler 调用新任务时，先 check_sleep 唤醒到时间的任务。

1.3 遇到的困难

（1）清完屏之后狂打 LPPPPPPPP，具体如图 1.

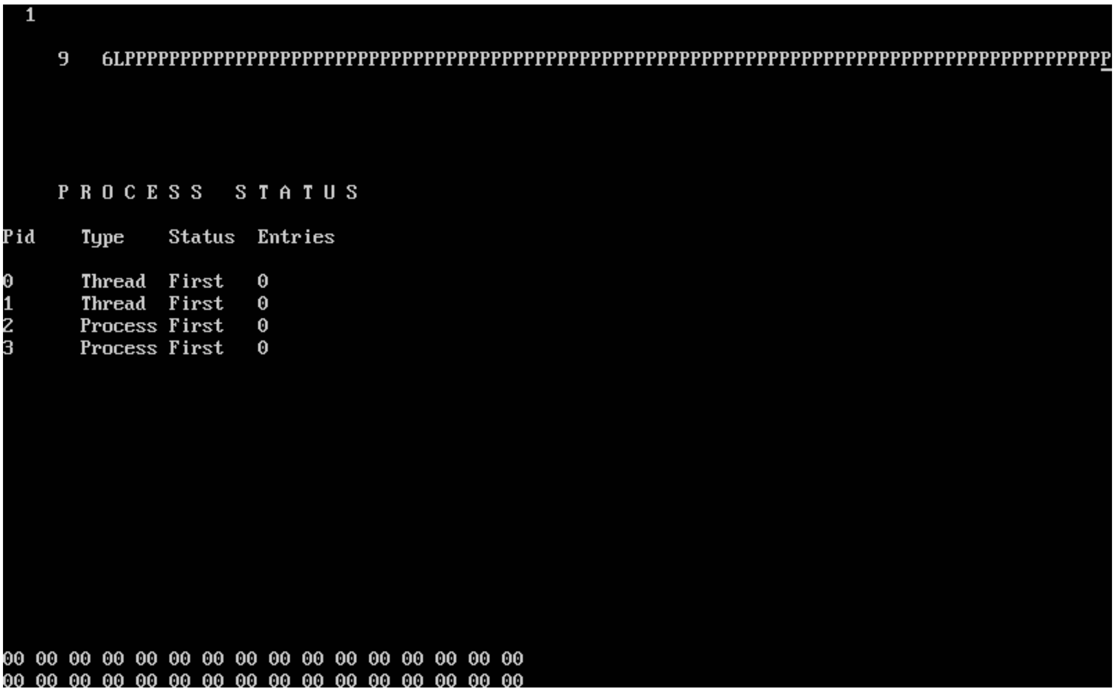

图 1. 错误 1

解决过程：最开始一直以为打 LPPPPP 是因为 LEAVE_CREATICAL 有问题，一直在找哪里多写了一个，并定位发现 scheduler 是执行了一遍的，查了一晚上没查出来哪错了。最后同学帮我查代码，发现我初始化每个任务的 PCB 时，29 号寄存器，即 sp，根本没有分配栈，意味着当执行任务的时候，sp 指针没有在自己的栈里跳，而是跑到了别的地址执行，就会出错。添加了栈之后就可以运行线程和进程了。

（2）process2 狂运行，th1 和 th2 只运行一次就不会再运行了，process1 根本没有运行，如图 2.

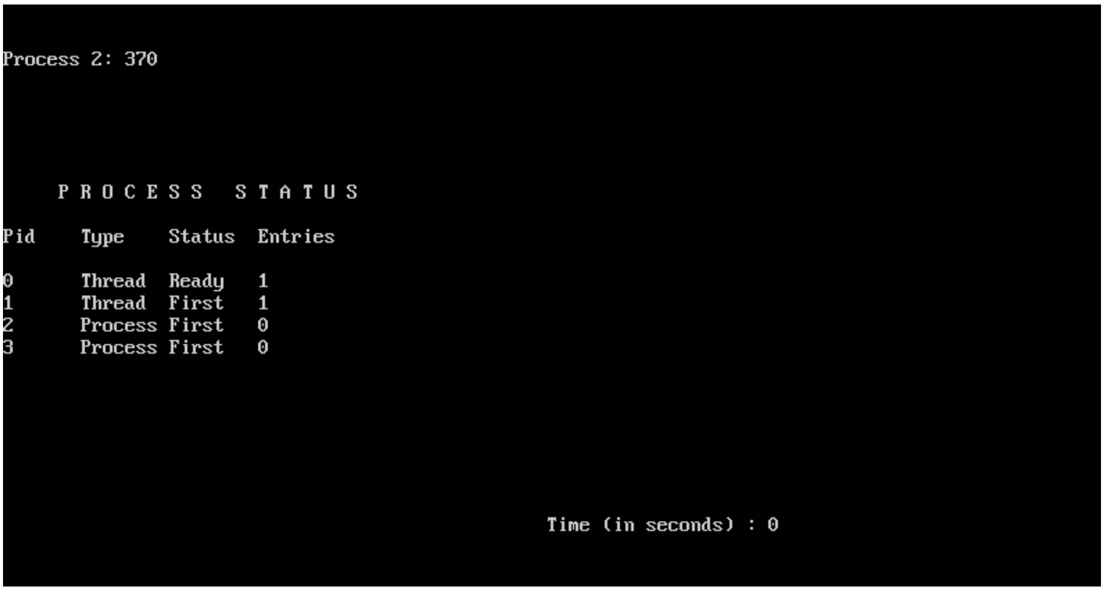

图 2. 错误 2

解决过程：p2 里的程序是死循环，如果不被时间中断打断就会一直运行，说明我的时间中断根本没有打断它。最后发现我的 PCB 初始化时没有把任务的特殊寄存器 CAUSE 的 IE 位和 IM7 位置 1，也就意味着当调度任务后，恢复现场，CAUSE 寄存器 IE 和 IM7 位都是 0，

根本就不允许时间中断。这个地方任务书没有写，我就忽视了，还是经同学指出才意识到的。但是，加了初始化之后仍然是 p2 一直在刷新示数，于是继续查错，发现 scheduler_entry 汇编代码最后一句没有加 jr ra。加上之后就能四个任务相继执行了。一直很奇怪为什么，直到室友加了 jr ra 还是没有进入中断程序，然后我们对照了一下代码发现她没有在 PCB 初始化时给 ra 寄存器赋值 entry_point，我们这才意识到，原来用到的是 ra 初始化的 entry_point 跑到任务的入口地址执行，而不是 PCB 结构体中 entry_point 域。

（3）四个任务都运行了但是 th1 和 th2 运行到第二次时 process2 会卡死，并且打印的数的光标会莫名跑到屏幕最底下。

解决过程：handle_int 的 SAVE_CONTEXT 位置不对。应该在刚进入中断就 SAVE，而我放到了真正中断处理的部分，即调度新任务的地方，在这中间，中断处理还做了一些工作，会导致再次恢复的时候根本不是中断停止的地方，就会跑偏并且错乱。

（4）测试 blocksleeping 的时候只有 th1 成功被唤醒，th2 一直没醒，就卡了，如图 3.

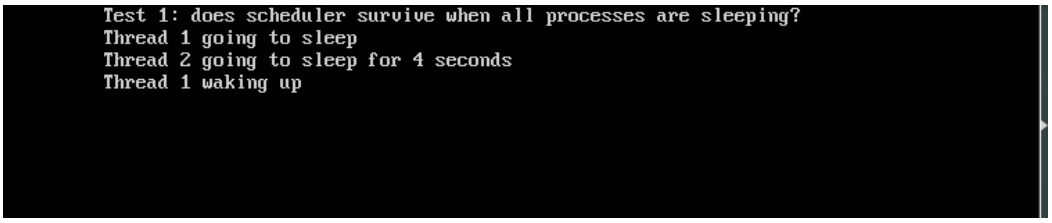

图 3. 错误 3

解决过程：在检查 sleeping 相关代码的时候发现 do_sleep 函数里，传入的参数和 time_elapsed 单位不一样，传入的参数应该除以 1000 再加当前时间作为 deadline，但我没有除，可能导致时间太长。另外又改了一下 check_sleeping 函数中，检查 deadline 的代码段，将所有到时间的任务都 dequeue 出来，这里注意，dequeue 传入的指针是要拿出的那个任务的前一个任务，即 prev 域指向的任务。

2. 基于优先级的调度器设计

2.1 优先级设置

在 PCB 结构体中设一个 Priority 域，用来存放每个任务的优先级，初始化时是将 pid 乘 10 作为初始优先级，这样四个任务的优先级分别是 10,20,30,40.每次使用 scheduler 调度任务时，先扫描队列找到优先级最大的，dequeue 出来该任务，并将其优先级减一，这是为了防止优先级最高的任务一直被执行。当扫描队列时发现有优先级变成 0 的再根据 pid 恢复成初始优先级。

2.2 遇到的困难

（1）由于代码顺序有问题导致一轮优先级都变成 0 后就会狂执行一个任务。

解决过程：发现恢复优先级最开始是对选出的任务做的，也就意味着将其恢复成初始化的优先级之后，队列里就会变成三个优先级 0 和一个超高优先级的，就会狂执行这个任务。为了防止这种现象出现，应该在所有优先级都变成 0 后整体恢复一遍，防止一个任务先恢复了，优先级太高。

3. 关键函数功能

4.1 scheduler_entry

是调度新任务的入口，负责保存并恢复上下文，调用 scheduler 函数对任务进行调度，并跳转到该任务执行。虽然代码只有短短的几行，但是缺了任何一行或者顺序错误都不对。正确代码如图 4.

```
221 NESTED(scheduler_entry,0,ra)
222     /* TODO: need add */
223
224   SAVE_CONTEXT(KERNEL)
225   jal     scheduler
226   nop
227   LEAVE_CRITICAL
228   RESTORE_CONTEXT(KERNEL)
229   STI
230
231   jr ra
232
233     /* TODO: end */
234 END(scheduler_entry)
```

图 4. Scheduler_entry 代码

4.2 handle_int

时间中断处理程序，用于判断中断信号是否为时钟中断，并进行处理。该函数是本次实验最重要的函数。具体实现的内容在上文已提过。需要注意的是在这里面进行任务调度前，要进行 enter_critical，这个地方正好与 scheduler_entry 函数中的 leave_critical 对应。代码如图 5.

```
jal   put_current_running
nop
jal   enter_critical
nop
jal   scheduler_entry
nop
#SAVE_CONTEXT(KERNEL)

RESTORE_CONTEXT(USER)
```

图 5. handle_int 函数部分代码

4.3 scheduler

调度新任务的程序，选出优先级最高的任务并作为 current_running. Scheduler 函数中最重要的代码段是扫描整个 ready 队列，找到优先级最高的任务，拿出来并将优先级减 1.代码如图 6.

```
    while(pcb != &ready_queue){
        temp = (pcb_t*)pcb;
        if(temp->priority > max_pri){
            best = temp;
            max_pri = temp->priority;
        }
        pcb = peek(pcb);
    }

    if(best->priority == 0){
        while(pcb != &ready_queue){
            temp = (pcb_t*)pcb;
            temp->priority = 10*(temp->pid);
            if(temp->priority > max_pri){
                best = temp;
                max_pri = temp->priority;
            }
            pcb = peek(pcb);
        }
    }

    best->priority = best->priority - 1;
}
```

图 6. Scheduler 重要代码段

■