Project3 Preemptive Kernel 设计文档

中国科学院大学
[王苑铮]
[2017.11.16]

1. 时钟中断与 blocking sleep 设计流程

（1） 中断处理的一般流程

1. 保存 user 上下文
2. 检查中断标志位，查看是哪种中断，跳到对应的中断处理程序。如果中断不需要处理（比如这次实验中，线程不会被时钟中断断开），则跳到 3.中断返回
3. 中断返回：将对应的中断标志位清零，恢复 user 上下文，开中断，中断返回

（2） 你所实现的时钟中断的处理流程，如何处理 blocking sleep 的 tasks；如何处理用户态 task 和内核态 task

我的时钟中断处理流程：

1. 时间++
2. 如果 current_running->nested_count == 0（用户态 task）则进入 3 继续处理，否则（内核态 task）直接返回
3. 软硬件关中断
4. 设置 current_running->nested_count =1
5. 当前 task 放入 ready 队列
6. 如果 current_running 是进程则 nested_count 置为 0，否则置为 1（我觉得按照课件上的流程，5 这一步应该放在 scheduler 或者 scheduler_entry 里，等进程下一次再被调度出来时再重新设置 nested_count，但是这样无法运行，会报 lppppp，只有放在这里才可以运行，很奇怪。）
7. 进入调度器 scheduler()

对 blocking sleep 的 task 的处理：
进入调度器 scheduler()后，第一步就是 check_sleep，遍历 sleep 队列把到达 sleep 时间的 task 从 sleep 队列弹出放入 ready 队列，等待后续被调度出来

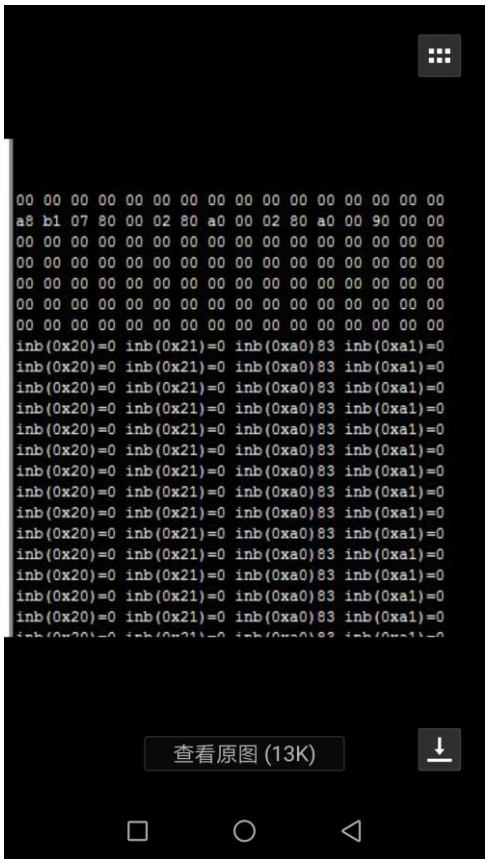
（3） blocking sleep 的含义，task 调用 blocking sleep 时做什么处理？什么时候唤醒 sleep 的 task？
Task 调用 do_sleep 把自己放入 sleep 队列，从当前时间起睡眠 n us。n us 过后，处理时钟中断时，调度器 check sleep 发现这个 task 的睡眠时间到头了，就把它从 sleep 队列弹出放入 ready 队列等待后续被调度出来执行

（4） 设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必

需项）

1. 重置 current_running->nested_count,我觉得按照课件上的流程,应该放在 scheduler 或者 scheduler_entry 里，等进程下一次再被调度出来时再重新设置 nested_count，但是这样无法运行，会报 lppppp，只有放在 timer_irq 里调用调度器 scheduler() 之前才可以运行。不知道是不是我的理解问题
2. 我的 task1 写完后上板会报如下的信息。和已经成功的小伙伴对比了一下，感觉我们的实现差不多。我后来试了下，拷了一份他的工程，然后把我的文件一个一个替换进去，每替换一个就上一次板，除了 entry.S 替换进去会出错，别的都不出错。之后，我又在他的 entry.S 里面，一个函数一个函数的替换成我自己的函数，每替换一个就上一次板，然后我上板就成功了。很无法理解。怀疑可能是我不小心把老师写好的某个函数不小心动了一下，导致无法运行。

   这个错误本来想发给蒋老师看看的，但是以前有 bug 的工程没有存档，现在被能正常运行的工程覆盖了



蒋老师说这种 bug 可能是返回地址有问题。可是我报这个 bug 是执行到 restore context 里面出现的，还没执行到到返回

2. 基于优先级的调度器设计

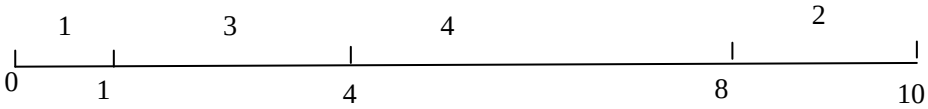
（1） priority-based scheduler 的设计思路，包括在你实现的调度策略中优先级是怎么定义的，如何给 task 赋予优先级，调度与测试用例如何体现优先级的差别通过随机数的方法进行调度。

优先级的定义：在一段时间上，权重对应于他被调度出来的次数（概率），权重高的被

调度出来的次数多，但是由于调度是基于随机数的，所以虽然优先级高的调度出来次数多，但顺序上未必是被先调度出来的。

优先级的赋值：在 kernel.c 里有个全局数组 weight_setting,依次是 task1,task2,task3….的权值。当然也可以设定成权值动态改变，每个 task 调度多次后可以降低它的权值，但我没有写

```
38  //int weight_table[3]={5,1,8};
39  int weight_setting[]={2,5,1,4,5,6,7,8,9,10};
40  int weight_table[NUM_TASKS];
41  int scheduler_table[NUM_TASKS];
42  int num_tasks = NUM_TASKS;
```



举例：有四个 task1，task2，task3，task4，权重分别赋值为 1,3,4,2,权重之和 total_weight=1+3+4+2=10

有一个数组存储每个 task 的权重，另一个数组存储这个 task 对应的随机数区间的上限（如上图的数轴所示）

调度：取一个随机数 rand, rand%total_weight 得到一个[0,total_weight-1]的随机数。看这个随机数落在数轴上哪个区间，将这个 task 调度出来。这样的话，取出一个 task 的概率就等于它的权重在总权重中的占比。当时间比较长时效果就比较明显了。（前提是随机数要比较随机）

如果扔出的随机数恰好那个 task 是 block、sleep、exit 的怎么办？我的设计是，继续扫描 task 找到下一个是 first 或者 ready 的调度出来。这样在一定程度上会破坏优先级，但是能调度出来一个任务至少比让 cpu 空等要好。而且即使如此调度，也依然是随机的，可能在宏观尺度上并不太影响按照权重调度的时间分配比例。

能避免空等并且不破坏优先级的做法，是动态维护优先级表，如果有 task sleep、block、exit，就把暂时无法调度的 task 去掉，更新数轴。但这样的问题是，如果 task 很多，更新数轴数组可能需要的时间比较长。因此，实际情况下，也许我们可以一定程度上允许违背优先级，换来比较快速的调度。

（2）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

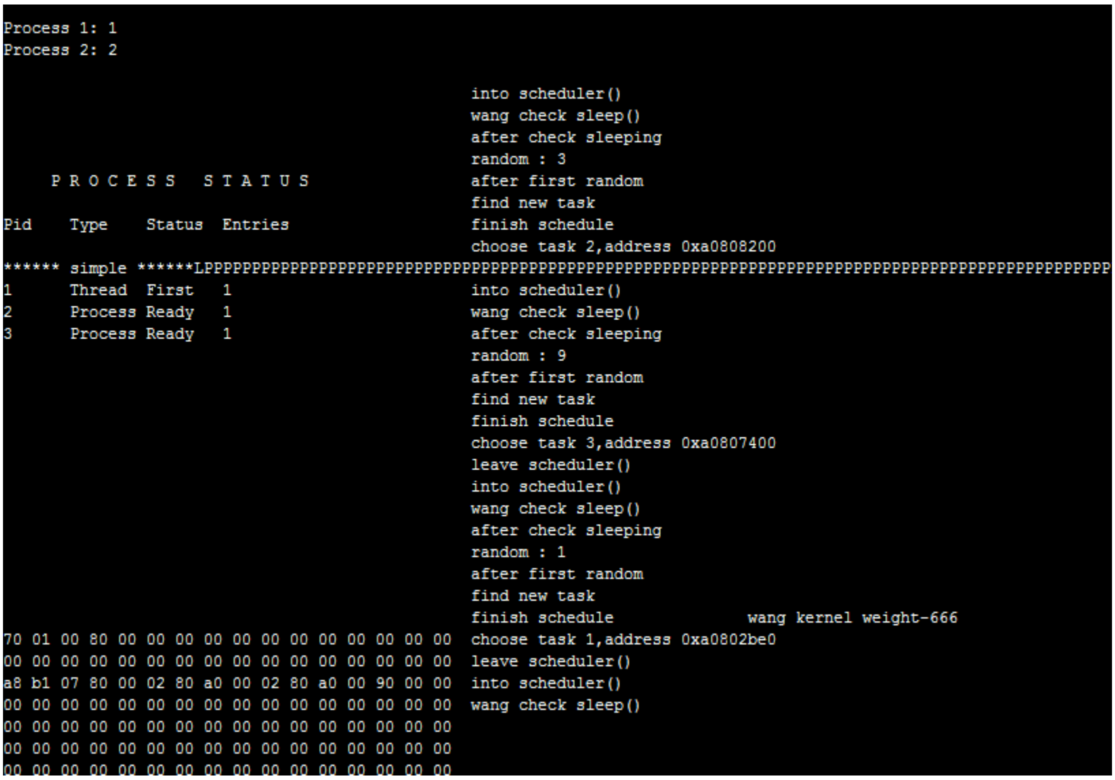
这个函数调试起来非常的无奈。。。。。

注：优先级调度的测试不涉及 sleep，为了缩小问题范围，我把 sleep 相关的东西都注释掉了

1. 一开始我调用的是 util.c 里的 rand(),但是它返回给我的随机数都是 0，不知道是不是我的用法有问题。于是我自己写了个随机数：用一个 create_rand.c 调用 c 自带的库函数 rand，生成了一千个随机数（一秒钟调度一次，十分钟 600 个随机数就够了），自动创建一个 myrand.h 头文件，里面是我的随机数数组。然后在 scheduler 里写了个 myrand()生成随机数，每次从 myrand 里的数组中读取下一个随机数。我的文件中，myrand.h 已经生成好了。运行 gcc create_rand.c 可以重新生成新的 myrand.h 替换掉旧的 myrand.h。我没有把 gcc create_rand.c 写在 makefile 里，所以每次 make 之后 myrand.h 不会变，这样每次随机出来的随机数其实是确定的，能比较方便的比较每次修改 kernel、scheduler 之后的结果

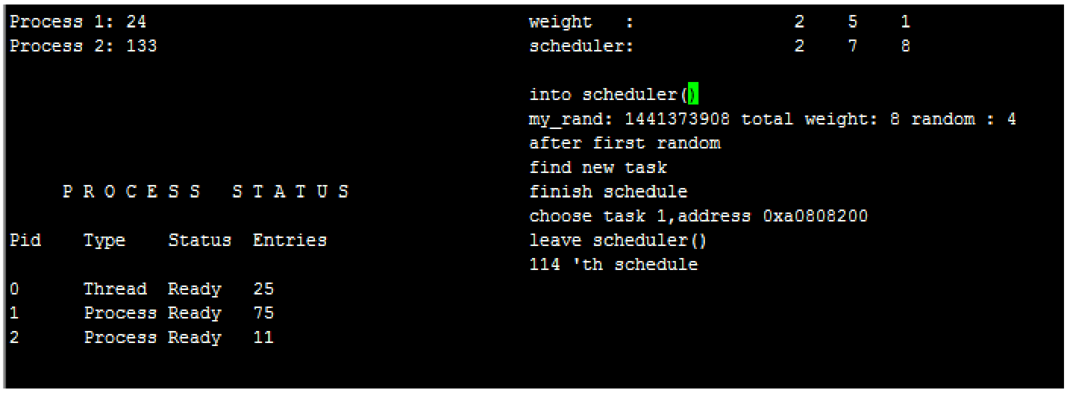
2. 当四个 task 的权值依次赋值为 1,2,3,4 时，只要调度出来第一个线程（打印时间的 clock），就会挂掉，报 lppppppppppppp。但是 tasks.c 中去掉这个线程后，就能正常运行。

权值 1,2,3,4，调度出线程 1 时就会打 lpppppppppppp



3. 在只有线程 2 和进程 1,2 时，如果线程 2 的权值比例比较高，当第二次调度线程 2（打印各个 task 的状态）时就会挂掉，打印 lpppppp。比如：设置为 1,3,2 或 2,5,1 都可以运行，但设置为 5,1,8 或 3,2,1 就会报 lpppppp

权值 2,5,1（时间长了会满屏打乱码，只能趁时间不长时截下来。但是因为次数不够多，调度比例不完全符合权重比例）：



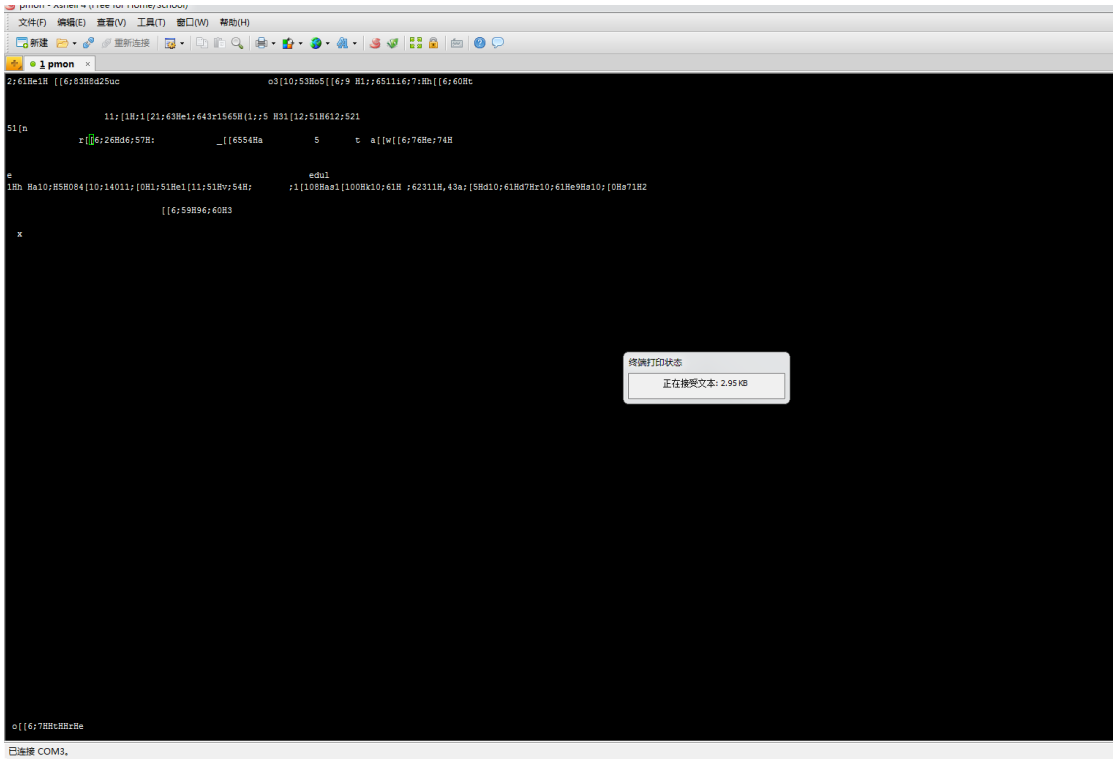
权值 1,3,2

```
Process 1: 140                          weight   :              1    3    2
Process 2: 178                          scheduler:             1    4    6

                                        into scheduler()
                                        my_rand: 2118085283 total weight: 6 random : 5
                                        after first random
                                        find new task
    P R O C E S S   S T A T U S         finish schedule
                                        choose task 2,address 0xa0807400
Pid    Type    Status  Entries          leave scheduler()
                                        210 'th schedule
0      Thread  Ready   34
1      Process Ready   103
2      Process Ready   71
```

权值 2,5,3

```
Process 1: 118                          weight   :              2    5    3
Process 2: 210                          scheduler:             2    7    10

                                        into scheduler()
                                        my_rand: 1068630273 total weight: 10 random : 3
                                        after first random
                                        find new task
    P R O C E S S   S T A T U S         finish schedule
                                        choose task 1,address 0xa0808200
Pid    Type    Status  Entries          leave scheduler()
                                        238 'th schedule
0      Thread  Ready   44
1      Process Ready   126
2      Process Ready   67
```

4. 在 scheduler 的结尾调用 print_status，即使原本能跑起来的（比如权值 1,2,3）也无法运行，上来就报 lpppp
5. 综上，打 lpppp 可能是 print_status 导致的。
6. 程序一段时间后就会满屏打乱码，甚至无法关闭 xshell，如图所示。不知道是程序的锅还是硬件的锅。



7. 强烈建议老师下次给个 mips 模拟器。上板 debug 只能靠打印信息，不能单步调试的话，根本无从下手。甚至有时候打印信息本身就会触发 lppppp。而且最好单步调试时能看见 c 语言而不是汇编。
   Pmon 的板子上是由单步调试的，但是很不好用，只能看见汇编。而且在调试模式下中断是被屏蔽的，所以和调度有关的 bug 也无法单步调试。
   这种只能看见现象却无法追踪原因的黑箱 debug 非常不合理。实验课的初衷应该是让我们能了解操作系统的基本架构，但是这种黑箱测试导致我们大量的时间花费在一些原因不明的 bug 上。我从昨天下午 2 点到晚上 9 点连饭都没吃在挑这个 bug 也没挑出来，周四有额外画了一个上午，也只是看见了现象，查不出原因。这种 debug 方式就算再给我一个星期也未必能查出来。

3. 关键函数功能

1. 中断处理函数：

```
NESTED(handle_int,0,sp)
  /* TODO: timer_irq */
  /* read int IP and handle clock interrupt or just call do_nothing */

  SAVE_CONTEXT(USER)
  mfc0  k0,   CP0_CAUSE
  andi  t6,   k0, 0x8000  #15th bit(start with 0th) IP7 means clock interupt
  beqz   t6,  1f       /* todo */
  nop

  #time irq
  #DEBUG(666)
  mtc0    zero, CP0_COUNT
  li      k1, 150000000
  mtc0    k1, CP0_COMPARE

  jal timer_irq
  nop

1:#interrupt return
  #DEBUG(667)
  mfc0  k0,   CP0_CAUSE
  li    k1,   0xffff00ff
  and   k0, k0, k1  #clear interrupt flags(in this lab,only time interrupt)
  mtc0  k0, CP0_CAUSE
  #DEBUG(668)
  RESTORE_CONTEXT(USER)
  STI
  #DEBUG(669)
  eret
  nop


  /* TODO:end */
END(handle_int)
```

2. 时钟中断处理函数

```
void reset_nested_count(){
    if(current_running->task_type == PROCESS)
        current_running->nested_count = 0;
    else
        current_running->nested_count = 1;

}



void timer_irq(){
    ++time_elapsed;
    //printstr("enter timer_irq()\n");
    if(current_running->nested_count == 0){
        enter_critical();
        current_running->nested_count = 1;
        put_current_running();
        reset_nested_count();
        //printstr("from timer_irq() to scheduler_entry()\n");
        scheduler_entry();
    }
}
```

3. do_sleep, check_sleep

```
/* TODO:wake up sleeping processes whose deadlines have passed */
void check_sleeping(){
    pcb_t* pcb;

  //  printstr("wang check sleep()\n");
    int task_num_in_sleep_queue = sleep_number;
    int i;
    for(i=0 ; i<task_num_in_sleep_queue ; ++i){
        pcb = (pcb_t*)dequeue(&sleep_wait_queue);
        //todo
        if(time_elapsed*1000 >= pcb->deadline){
            pcb->status = READY;
            enqueue(&ready_queue,(node_t*)pcb);
            --sleep_number;
        }
        else{
            enqueue(&sleep_wait_queue,(node_t*)pcb);
        }
    }
}
```

```
void do_sleep(int milliseconds){
      ASSERT(!disable_count);

      enter_critical();
      // TODO
         current_running->status = SLEEPING;
         ++sleep_number;
         current_running->deadline = get_timer()*1000 + milliseconds;
         enqueue(&sleep_wait_queue,(node_t*)current_running);
         scheduler_entry();
}
```

非优先级调度的 scheduler()是老师写好的，我就不贴上来的

4. 优先级调度：

   Scheduler():

```
/* Change current_running to the next task */
void scheduler(){
    ASSERT(disable_count);
    printf(5,50,"into scheduler()\n");
  /*   check_sleeping(); // wake up sleeping processes
    while (is_empty(&ready_queue)){
        leave_critical();
        enter_critical();
        check_sleeping();
    }
    */
//     current_running = (pcb_t *) dequeue(&ready_queue);

   //  printf(print_row++,50,"after check sleeping\n");

   //weight based scheduler
    int total_weight = scheduler_table[num_tasks-1];
    int my_rand = myrand();
    int random = my_rand % total_weight;
    int could_sheduler = 0;
    int new_task;

  //   printf(6, 50, "                                                  ");
    printf(6, 50, "my_rand: %d total weight: %d random : %d     \n",my_rand,total_weight,random);

    for(new_task=0; new_task < num_tasks ; ++new_task){
       if( scheduler_table[new_task] <= random)
          continue;
       else{
          could_sheduler = (pcb[new_task].status == READY || pcb[new_task].status == FIRST_TIME)?1:0;
          break;
       }
    }

    printf(7, 50,"after first random\n");
```

```
128     if(!could_sheduler){
129        for( ;  ; new_task = (new_task+1)%total_weight){
130            could_sheduler = (pcb[new_task].status == READY || pcb[new_task].status == FIRST_TIME)?1:0;
131            if(could_sheduler)
132                break;
133            else
134                continue;
135        }
136     }
137
138     printf(8, 50,"find new task\n");
139
140     dequeue(&ready_queue);
141     current_running = &pcb[new_task];
142
143     printf(9, 50,"finish schedule\n");
144     printf(10, 50,"choose task %d,address 0x%x\n",new_task,current_running->entry_point );
145     ASSERT(NULL != current_running);
146     ++current_running->entry_count;
147
148 //     print_status();
149
150     printf(11, 50,"leave scheduler()\n");
151     printf(12, 50,"%d 'th schedule\n",++schedule_time);
152
153 }
```

Kernel 中优先级的赋值：

```
38  //int weight_table[3]={5,1,8};
39  int weight_setting[]={2,5,1,4,5,6,7,8,9,10};
40  int weight_table[NUM_TASKS];
41  int scheduler_table[NUM_TASKS];
42  int num_tasks = NUM_TASKS;
```

```
71      //give weight
72      for(i=0; i < NUM_TASKS; ++i){
73         weight_table[i]=weight_setting[i];
74         scheduler_table[i] = (i==0)? weight_table[i] : scheduler_table[i-1] + weight_table[i];
75      }
76
```

参考文献

[1] [单击此处键入参考文献内容]

■