Project3 Preemptive Kernel 设计文档

中国科学院大学
李云志
2017.11.13

1. 时钟中断与 blocking sleep 设计流程

（1） 中断处理的一般流程

a) 保存被中断处理函数打断的进程的上下文
b) 确定中断类型
c) 根据中断类型跳转到相应的中断处理函数处
d) 处理完中断后，清中断并恢复被打断进程的上下文
e) 从中断处理函数返回正常任务流

（2） 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

a) 时钟中断的处理流程
   i. time_elapsed 变量值增加 1000，用于记录系统时间
   ii. 判断当前任务所处状态，是内核态则表明是内核线程，直接执行中断返回相关操作。如果是用户态进程，则先将其状态改为内核态后存入 ready 队列，再跳转至 scheduler 进行相关调度操作。
   iii. 获取到新的 task 后，恢复上下文，清中断，恢复正常的任务流
b) 如何处理 blocking sleep 的 tasks？

   当 task 调用 do_sleep 时表明其要进入休眠模式，则根据传入的参数以及系统当前的时间计算其 deadline，并将其加入到睡眠队列中。需要注意的是，在这地方并不是简单的加入到睡眠队列，而是有一个排序操作。这方便我们以后的唤醒操作。

c) 如何处理用户态 task 和内核态 task？

在此次任务中，时钟中断不会打断内核态的 task。所以原则上，在内核态的 task 在时钟中断到来的时候只需要进行一次保存上下文与恢复上下文。

（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 函数表明其要进行休眠。do_sleep 函数先关闭中断，并进行睡眠时间的计算，之后将 task 加入到睡眠队列中。如上一问所述，这里加入的队列为优先队列，唤醒时间早的位于前部。之后跳转到调度器，进行调度工作。

每次任务调度时，后会进行 checking_sleep 的操作，判断睡眠队列中的任务是否需要被唤醒。被唤醒的任务加入到 ready 队列中。

2. 基于优先级的调度器设计

（1）priority-based scheduler 的设计思路

a) 本此实验，采用的优先级设计如下：
   优先级分为 0-9 级，其中 9 级最高。
b) 调度器调度策略如下：

将 ready 队列中当前优先级最高的任务取出并执行。每执行一次，任务的优先级减 1.

c) 在任务执行第一次时初始化优先级

d) 若 ready 队列中所最高优先级的任务的优先级为负数时，初始化所有任务的优先级。

3. 关键函数功能

1、 判断中断类型：

```
mfc0  k0, CP0_CAUSE
nop
andi  k0, k0, CAUSE_IPL
clz   k0, k0
xori  k0, k0, 0x17
addiu k1, zero, 7
beq   k0, k1, time_irq
nop
jal   clear_int
nop
```

2、 时钟中断处理函数：

```
time_irq:
  li    a0, TIMER_INTERVAL
  jal   reset_timer
  jal   clear_int
  lw    k1, time_elapsed
  addiu k1, k1, 1000
  sw    k1, time_elapsed
  TEST_NESTED_COUNT
  bne   k1, zero, int_done
  nop
  ENTER_CRITICAL
  lw    k0, current_running
  li    k1, 1
  sw    k1, NESTED_COUNT(k0)
  jal   put_current_running
  nop
  jal   scheduler_entry
  nop
  lw    k0, current_running
  sw    zero, NESTED_COUNT(k0)
  LEAVE_CRITICAL
  j     int_done
  nop
```

3、 do_sleep：

```
void do_sleep(int milliseconds){
     ASSERT(!disable_count);
     enter_critical();
     // TODO
     current_running->status = SLEEPING;
     current_running->deadline = do_gettimeofday() + milliseconds;
     sort_and_enqueue(&sleep_wait_queue, (node_t *) current_running, lte_deadline);
     scheduler_entry();
     leave_critical();
}
```

4、 checking_sleep：

```
void check_sleeping(){
  uint32_t now = do_gettimeofday();
  node_t *tmp;
  while(!is_empty(&sleep_wait_queue)&&((pcb_t*)peek(&sleep_wait_queue))->deadline <= now)
  {
    tmp = dequeue(&sleep_wait_queue);
    enqueue(&ready_queue,tmp);
    ((pcb_t *)tmp)->status = READY;
  }
}
```

5、 自实现的优先级调度的 scheduler：

```
void scheduler(){
     ASSERT(disable_count);
     check_sleeping(); // wake up sleeping processes
     while (is_empty(&ready_queue)){
          leave_critical();
          enter_critical();
          check_sleeping();
     }
     current_running = (pcb_t *) dequeue(&ready_queue);
     pcb_t *record = current_running;
     pcb_t *temp = record;
     do{
       enqueue(&ready_queue,(node_t*)current_running);
       current_running = (pcb_t*)dequeue(&ready_queue);
       if(running_priority[current_running->pid]>running_priority[temp->pid])
       {
         temp = current_running;
       }
     }while(current_running!=record);
     if(running_priority[temp->pid]<0)//刷新所有的优先级
     {
       do{
         enqueue(&ready_queue,(node_t*)current_running);
         current_running = (pcb_t*)dequeue(&ready_queue);
         running_priority[current_running->pid] = original_priority[current_running->pid];
       }while(record!=current_running);

     record = current_running;
     temp = record;
      do{
       enqueue(&ready_queue,(node_t*)current_running);
       current_running = (pcb_t*)dequeue(&ready_queue);
       if(running_priority[current_running->pid]>running_priority[temp->pid])
       {
         temp = current_running;
       }
     }while(current_running!=record);
     }
     if (current_running!=temp){    //维护队列，弹出优先级最高的任务
       //enqueue(&ready_queue,(node_t*)current_running);
       temp->node.prev->next = (node_t*)current_running;
       temp->node.next->prev = (node_t*)current_running;
       current_running->node.prev = temp->node.prev;
       current_running->node.next = temp->node.next;
       current_running = temp;
     }
     running_priority[current_running->pid]--;
     ASSERT(NULL != current_running);
     ++current_running->entry_count;
}
```

参考文献

无

■