Project2 A Simple Kernel 设计文档（Part II）

中国科学院大学
蔡润泽
2019 年 10 月 13 日

1. 时钟中断、系统调用与 blocking sleep 设计流程

（1） 时钟中断处理的流程，请说明你认为的关键步骤即可

1）在初始化操作完成后，当 cp0_compare==cp0_status 时，硬件自动跳转至 0x80000180 处，此处有之前在初始化阶段搬运过来的 exception_handler_entry。exception_handler_entry 负责保存用户态内容，并判断例外的类型，此处例外类型为中断，因此系统跳转至中断处理程序 handle_int。

2）中断处理程序 handle_int 负责取出 CP0_STATUS 和 CP0_CAUSE 的值，并借助 interrupt_helper 程序判断中断类型，此处终端类型为时钟中断。因此，系统跳转至时钟中断处理程序。

3）时钟中断处理程序负责累加运行时间，并利用 do_scheduler 程序，完成上下文切换。

4）上下文切换完成后，进入新进程的用户态，并恢复用户态内容。

（2） 你所实现的时钟中断的处理流程中，何时唤醒 sleep 的任务？

在进行上下文切换时，scheduler 函数首先会利用 check_sleep 函数来检查阻塞队列中是否有 sleep 时间已达到的进程。利用 get_timer 函数，该操作系统可以获取当前时间，将当前时间与进程起始睡眠时间相减，若结果大于等于睡眠时间，则该进程被唤醒，进入到就绪队列。

（3） 你实现的时钟中断处理流程和系统调用处理流程有什么相同步骤，有什么不同步骤？

相同步骤：硬件自动跳转至 0x80000180 处，此处有之前在初始化阶段搬运过来的 exception_handler_entry。exception_handler_entry 负责保存用户态内容，并判断例外的类型。当系统调用处理完毕后，恢复用户态内容。

不同步骤：

1）触发系统调用的函数是利用 syscall 指令。

2）系统调用处理函数是借助初始化后的 syscall 表来完成的。

（4） 设计、实现或调试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）

这部分实验主要遇到了两个问题：

1）cp0_status 初始化的问题。在运行 task3 时，我的 cp0_status 寄存器初始化只置为了 0x10008001 而非 0x10008003。这个设置在运行 task3 的时候还可以进行，但实验运行到 task4 时，就出现了问题。根据老师上课的提示，我将其初值置为了 0x10008003，系统停止运行的 bug 消失。

2）打印输出的问题。我发现本次实验中，printf 的输出会重叠在之前的输出内容上，导致屏幕出现残影。我通过设置空格和更改 print_location 的值解决了这一问题。

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，测试时给不同任务赋的优先级是多少，结果如何体现优先级的差别

在优先级设置时，该设计综合考虑了两个因素：①任务本身的优先级②等待时间的优先级。进程最开始会按照相应任务本身的优先级进入就绪队列。当一个任务被执行时，就绪队列中其他任务的优先级会增加 1，同时当前进程的优先级被还原成任务优先级。同时，对于被阻塞的队列，也会根据等待时间的长短，依次增加其优先级的值。

在任务优先级方面，不同任务的任务优先级如下：（结构中第三个元素为任务优先级的值）

```
1.  // test_lock2.c : User space lock test
2.  struct task_info task2_4 = {(uint32_t)&lock_task1, KERNEL_THREAD,10};
3.  struct task_info task2_5 = {(uint32_t)&lock_task2, KERNEL_THREAD,10};
4.
5.  /* [TASK4] task group to test interrupt */
6.  // When the task is running, please implement the following system call :
7.  // (1) sys_sleep()
8.  // (2) sys_move_cursor()
9.  // (3) sys_write()
10. struct task_info task2_6 = {(uint32_t)&sleep_task, USER_PROCESS,5};
11. struct task_info task2_7 = {(uint32_t)&timer_task, USER_PROCESS,5};
12.
13. struct task_info task2_8 = {(uint32_t)&printf_task1, USER_PROCESS,40};
14. struct task_info task2_9 = {(uint32_t)&printf_task2, USER_PROCESS,40};
15. struct task_info task2_10 = {(uint32_t)&drawing_task2, USER_PROCESS,40};
```

3. Bonus 设计思路（做 bonus 的同学需要写该节）

（1） Bonus 设计

1） 将 block_queue 队列由 1 个更改为 NUM_MAX_TASK 个，并将 lock 的数据结构增加 id 值。

2） 在锁初始化时，将 lock->id 依次增加，锁的申请和释放的阻塞队列编号，对应 lock->id 的值

（2） 测试用例

本次实验中，我增加了一个 lock_test3 文件，利用三个任务，抢占两个锁。任务一先抢占锁 1，后抢占锁 2；任务而先抢占锁 2，后抢占锁 1。任务三只抢占锁 1。但由于进行本次实验的时间较紧，lock_id 的初始化值出现了不为 0 的现象，需要在后续更改 createimage 来修复这一问题。

4. 关键函数功能

请列出你觉得重要的代码片段、函数或模块（可以是开发的重要功能，也可以是调试时

遇到问题的片段/函数/模块）

重要函数片段：

本次 PCB 初始化中，对于 ra 寄存器等设置：

```
1.  pcb[queue_id].kernel_context.regs[31]= (uint32_t)reset_cp0;
2.  pcb[queue_id].kernel_context.cp0_epc = lock_tasks[i]->entry_point;
3.  pcb[queue_id].kernel_context.cp0_status=0x10008003;
```

prioity_queue_push（）的根据优先级将任务置入就绪队列的操作。

参考文献

[1] project_2_partII_guide_book