任务一：时钟中断处理和 blocking sleep 实现

中国科学院大学 操作系统研讨课

2017.10.19

1. 介绍

本次作业的目的是了解抢占调度在内核中的实现以及运行方式。与作业二不同，系统调用不再使用伪系统调用且 task（进程）在运行过程中被时钟中断，进而执行调度操作。

本次任务需要实现一个时钟中断处理器，以及阻塞睡眠操作，即将当前进程阻塞，并睡眠，然后调度其它进程。

1.1. 需要了解的部分

中断的处理流程，时钟中断的处理操作

进程睡眠时如何处理，如何唤醒进程。

1.2. 注意

task 通过时钟中断器进行抢占调度。在 MIPS 架构中，软件、硬件、异常都可以进入同一个异常处理入口，然后根据 CP0 CAUSE 寄存器区分究竟是异常还是中断，分别进行处理 handle_int 和 handle_syscall。MIPS 通过 CAUSE 寄存器的 IP7 bit 位向 CPU 报告时钟中断，因此，软件中断分发处理代码需要根据 IP7 的值判断时钟中断，进而调用对应的中断处理程序。

2. 初始代码

2.1. 文件介绍

- Makefile: 编译文件。
- bootblock.s: 内核启动程序，**请使用作业一中自己写的代码**。
- Createimage.c: 生成内核镜像的Linux工具，**请使用作业一中自己写的代码**。
- entry.S: 时钟中断处理函数，在本次任务中需要完成handle_int和scheduler_entry函数，以及TEST_NESTED_COUNT、SAVE_CONTEXT和RESTORE_CONTEXT宏。
- kernel.c: 内核最先执行的文件，放在内核的起始处，需要完成任务的初始化操作。
- interrupt.c: 系统调用和中断处理相关的函数。
- scheduler.c: 调度器，实现*task*的调度，本次任务需要对调度进行完善。
- syslib.S: 系统调用函数，进程通过系统调用进入内核，本次任务需要了解。
- queue.c: 队列处理函数，提供了队列操作的一些接口。
- print*.c：提供一些输出函数，可以用于调试以及显示信息。请在本任务开始前了解该文件。
- sync.c: 一些同步操作。
- util.c: 提供了一些输出函数，可以用于调试以及显示信息。请在本任务开始前了解该文件。
- settest: 设置需要测试的样例。
- test_*文件夹：针对不同任务提供不同的测试，本次作业需要测试test_preemp和test_blocksleep。
- *.h: 相应.c/.S文件的头文件。

2.2. 获取：

课程网站。

2.3. 运行

Makefile 文件提供编译功能。

./settest test_XXX 设置测试对象以及编译

make 编译命令

make clean 对编译产生的文件进行清除

sudo dd if=image of=/dev/sdb 将产生的 image 写进 SD 卡中

在 minicom 中执行 *loadboot* 运行程序

2.4. 注意

初始代码中包括部分系统调用、异常处理器，在实现时钟中断处理之前，可以先参考系统调用以及其他处理器的实现方式。

作业开始前可以先参考 scheduler.h 和 queue.[ch]提供的已经实现的接口。

中断处理的步骤：STATUS 寄存器末位清零关中断；通过 CAUSE 寄存器判断是否是时钟中断；进入异常/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；读取寄存器，恢复现场；STATUS 寄存器末位置 1，开中断。

3. 任务

3.1. 设计和评审

帮助学生发现设计的错误，及时完成任务。学生需要对这次的作业进行全面考虑，在实现代码之前有清晰的思路。学生讲解设计思路时可以用不同的形式，如伪代码、流程图等，建议使用 PPT。

3.1.1. 设计介绍

中断处理流程，可以参考系统调用处理流程。

时钟中断处理流程。

Blocking sleep:

- Task调用睡眠时做什么处理，何时、怎样唤醒一个task？
- 当就绪队列为空时，即所有的task都在睡眠，此时你应该怎么做？

3.2. 开发

这次任务分三个部分：

- 中断的处理以及时钟中断的处理流程
- blocking sleep,实现scheduler.c中的check_sleeping(), put_current_running()和do_sleep()函数
- 任务初始化，将作业二做的操作扩展到作业三代码中，包括任务初始化，保存pcb等。本作业中pcb结构体的定义已经提供，其中需要保存两部分信息，一部分为用户态信息，一部分为内核信息。

3.2.1.要求

在产生中断时(handle_int: entry.S)：

1. 保存中断上下文，即SAVE_CONTEXT(USER)，保存进程的用户态信息。
2. 读取引起中断的IP bit位，判断中断类型（参考课件），若为时钟中断则跳到时钟中断处理函数中，否则执行清中断（跳到4进行处理），返回原进程继续执行。
3. 在时钟中断处理函数(timer_irq)中，需要做如下工作：
   1) 关中断(enter_critical)。关硬件中断以及增加disable_count: 来表明中断已经被屏蔽。
   2) 增加计数器:time_elapsed，用于blocking sleep。
   3) 检查nested_count是否为0（用TEST_NESTED_COUNT宏实现）。如果为0，需要首先进入内核（1代表在内核态，0代表在用户态），再进行处理，否则时钟中断处理函数直接返回，即跳到4进行处理
   4) 将当前进程加入到就绪队列中。
   5) 使用scheduler_entry()函数调度新的任务。注意：调度时需要保存进程的内核态信息，在代码中为SAVE_CONTEXT(KERNEL)，KERNEL是SAVE_CONTEXT宏的参数，表明当前保存内核态
4. 中断处理函数返回时，需要清中断，即把CAUSE寄存器IP域中不为0的

最高位清零，在本任务中清除的是IP7 bit位（对应时钟中断）。注意：此处可以借助于clz指令，该指令用途：count the number of leading zeros in a word，即由最高位到最低位找到第一个为1的位的位置。

5. 开中断
6. 恢复上下文（RESTORE_CONTEXT）

注意：本任务中 entry.S 中的 SAVE_CONTEXT 操作，和作业二中的 SAVE_PCB 实现相同的功能，但是实现方式不一样。SAVE_CONTEXT 使用宏，后面带一个参数用来指定保护用户态或者内核态信息。作业二中的 PCB 保存是基于函数调用的（ra 寄存器），且一个栈就可以。本次任务中的 PCB 保存是由中断或者系统调用引起的，中断时可能没发生函数调用，因此需要分别保存用户态信息和内核态信息。当系统调用或中断产生时，先保存用户态信息。当在内核中进行调度时（例如 scheduler_entry），需要保存内核信息。此外，在作业二的 PCB 保存基础上仍需要做：保存通用寄存器和特殊寄存器，其中特殊寄存器包括 status、cause、epc、badvaddr、lo、hi。这些寄存器主要用于判断中断和异常产生的原因以及相关地址。

对于 scheduler_entry 操作，需要在调用 scheduler 函数前保存内核上下文 SAVE_CONTEXT(KERNEL)，调度后恢复上下文 RESTORE_CONTEXT(KERNEL)

对于关中断和开中断。CLI 和 SLI 宏在硬件上执行关中断和开中断。ENTER_CRITICAL 和 LEAVE_CRITICAL 宏在软件上设置了一个 disable_count 变量，用来判断此时关/开中断是否正确，如果不正确会执行 panic 操作。

Blooking sleep:

1. check_sleeping()：唤醒那些当前时间已经达到所需运行时间的task。
2. put_curring_running(): 保存当前进程。
3. do_sleep(): 当前进程被阻塞，调用下一个需要运行的进程。

3.2.2.注意事项

pcb_t 结构体本任务已经提供，user_tf 保存用户态信息，kernel_tf 保存内核态信息。其中 trapframe_t 中的 regs[32]数组用来表示 32 位通用寄存器，从寄存器 0 到 31，剩余几位变量的名字和特殊寄存器相对应。system_call_helper 函数对这些寄存器有操作，除非你有充分把握，不要轻易修改 pcb_t 结构中寄存器的位置，若修改，需对应修改 system_call_help 函数。

nested_count 是 PCB 结构体中的一个域，用来指定是否在内核态。对于线

程来说，则总是 1；对于进程，在内核态时需要为 1，其它为 0，所以进程在系统调用时设置为 1。中断处理需要在内核态进行，所以需要 nested_count 设置为 1。TEST_NESTED_COUNT 宏用来检查 nested_count 是否为 1，即是否有处理中断的权限。如果检查 nested_count 为 0，即发生中断时进程在用户态，此时需要进程先进入内核态，即设置为 1, 在进程返回用户态时再设置为 0。

在屏蔽中断的问题上，需要满足两个条件：

- 安全性：当你的代码正在访问临界区（即内核的数据结构，主要为PCB和任务队列）时，需要关闭中断。
- 灵活性：中端被屏蔽的时间应尽可能短，避免进程占用CPU的时间太长。

计数器 time_elapsed（scheduler.c）用来记录时钟中断发生的次数。板子主频是 300MHZ，COUNT 寄存器两个 cycle 增 1。等到下次这两个寄存器的值相等，才会触发下一次中断。也就是说，1 秒钟定时器自动触发一次中断。

4. 测试

在每次测试之前，运行./settest xxxx, 配置需要测试的对象。

测试进程抢占的正确性：

- 测量Test_preempt: 创建两个进程，两个进程各自不断循环，但是没有调用yield和sleep,如果看到两个进程的计数器值都在增加，则说明抢占器实现成功。但是有可能两个进程的计数器的差值比较大，这是在一定误差范围之内的。
- 测量blocking sleep: Test_blocksleep, 创建两个线程，并且一个是睡眠状态。测试以下四个部分：当所有task都睡眠时的处理；进程在睡眠期间是否被执行；进程睡眠时长是否满足要求；非睡眠进程是否被调度。