任务二：同步原语实现

中国科学院大学 操作系统研讨课

2017.11.15

1. 介绍

本次任务需要实现三种同步原语。

1.1. 需要了解的部分

condition variables（条件变量）, semaphores（信号量） 和 barriers（屏障）的原理。

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。
- kernel.c: 内核最先执行的文件，放在内核的起始处。
- scheduler.c: 调度器，实现*task*的调度。
- syslib.S: 系统调用函数
- syslib.c: 系统调用接口
- interrupt.c: 系统调用和中断处理相关的函数。

- 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: 同步操作，本任务需要实现。
- mbox.c: 邮箱的操作，本次任务暂不需要实现。
- 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。
- file.c: 将所有任务放在File的结构体中，供do_spawn查找使用
- ramdisk.c：提供操作File的一些接口
- 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_barrier和test_all。
- *.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

需要在 sync.c 中完成同步原语的实现，在 sync.h 中完成同步原语结构的定义。

代码中已经给了一个锁的实现，可以进行参考。

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

- 对于condition variables, semaphore, and barriers三种同步原语，讲解其实现机制，以及本次任务 设计的数据结构。
- 分别讲解condition_warit, semaphore_up, semaphore_down和barrier_wait函数的实现方法。

3.2. 开发

实现三个同步原语，即 condition variables, semaphore 和 barriers。

3.2.1. 要求

实现以下函数：

Condition Variables:

- condition_init(void): 初始化condition variable的数据。
- condition_wait(lock,cond): 释放锁，阻塞任务，直到被唤醒，获得锁。调用task首先需要获得锁。
- condition_signal(cond): 释放condition_wait(lock,cond)第一个阻塞的task，如果没有task被阻塞，则什么都不做。
- condition_broadcast(cond): 释放condition_wait(lock,cond)组所的所有task。释放顺序可以不考虑。

Semaphore

- semaphore_init(sem, value):初始化semaphore的数据，初始化值为value,一个非负整数。
- semaphore_down(sem)：如果semaphore不是正数，则阻塞，否则减去1.
- Semaphore_up(sem)：semaphore增加1.

Barriers:

- Barrier_init(bar,n)： 初始化barrier有n个task。
- Barrier_wait(bar): 一直阻塞到有n个task调用barrier_wait(bar)。Task被阻塞的顺序不需要考虑。在第n个task调用之后立即返回，然后开始等待新的n个task.

3.2.2.注意事项

通过实现开和关中断来保证同步原语在时钟中断中正确执行。开关中断可以用 enter_critical 和 leave_critical。

所有数据需要先初始化在操作。如果操作发生在未初始化的数据中，结果取决于你的实现方式。阻塞线程就绪后都需要加入到就绪队列的末尾。

Semaphore 需要保证 semaphore_up 和 semaphore_down 成功的次数一样多，即最后 semaphore 的值为初始化的值。

4. 测试

在每次测试之前，运行./settest test_name_here, 配置测试。

测试 barriers 的正确性：test_barrier,创造三个线程，每个线程睡眠一个随机的时间，然后 barrier_wait 在同一个 barrier,此过程会重复几次。

测试所有实现的正确性：test_all, 测试本次任务所有的实现，如 lock，condition variables, semaphore, barriers 等。