第二次实例分析

注意

本次实例分析使用的xv6源码版本为rev11，linux的源码版本为4.12.10。

阅读须知：请配合提出的问题，针对性的阅读相关源码，忽略与本次实例分析无关的内容。**请不要做深度优先搜索！**

参考资料：

1. xv6配套讲义
2. i386手册

第一部分

自旋锁

相关文件：

spinlock.h、spinlock.c、x86.h

重点关注问题：

1. xv6中自旋锁 `spinlock` 的数据结构以及重要字段的作用
2. `acquire` 函数获取锁的时候，为什么首先进行要执行 `pushcli` “关中断”操作?
3. 内联汇编函数 `xchg` 返回 `lk->locked` 在 `xchg` 执行之前的值，考虑两个core上的分别有一个进程先后执行 `xchg` ，返回值分别为0和1时，这两个进程分别发生了什么？结合这个例子，说明xchg指令是原子指令的重要性。
4. 在编译后的xv6源码目录下执行 `objdump -d spinlock.o` ，观察 `acquir` 函数中， `xchg` 函数和它周围的while循环被编译成了什么汇编代码？试将该汇编代码和C代码对应起来。
5. 查阅内联汇编相关文档以及i386手册，结合objdump的输出，解释内联汇编函数 `xchg` (0568~0679)。
6. `acquire` 和 `release` 函数中的 `__sync_synchronize();` 语句的作用是什么？ 在汇编程序中是如何体现的?
7. 查阅i386手册，了解 `pushcli` 函数中的 `EFLAGS` 和 `FL_IF` 分别是什么？ `sti` 和 `cli` 内联汇编函数是如何影响前者的?
8. `mycpu()->intena` 变量的作用是什么?

Linux 原子操作与自旋锁

1. Linux为我们提供了一些原子操作的接口，即`atomic_t`类型的原子变量 [include/linux/types.h] 和相应操作 [arch/[arm\x86]/include/asm/atomic.h]，它的实现依赖于不同的体系结构，以下题目请以ARM32或者x86架构为例：
    - 查找上述代码中，有关`atomic_t`的数据结构以及相应的操作；
    - 分析`atomic_add`函数，主要说明通过什么样的硬件支持实现了原子操作。
2. Linux在内核版本2.6.25之后，实现了一套名为"FIFO ticked-based"算法的自旋锁机制，请通过阅读相关内核代码 [include\linux\spinlock_types.h, arch/arm/include/asm/spinlock_types.h] 以及提供的材料链接，给出相应的数据结构及重要字段的作用，并要求详细介绍这个机制的产生原因和具体原理。（之所以给出arm arch的ticket spinlock实现，是因为相比x86 arch下的实现要更为简洁直观，易于理解）

第二部分

条件变量

相关文件：

proc.c

重点关注问题：

1. 请查阅资料，给出`pthread`提供的条件变量操作，并与xv6提供的`sleep`&`wakeup`操作比较。
2. 为什么`sleep`需要一个`lk`作为参数？（可以参照xv6配套讲义中**sleep and wakeup**一节中的代码改进过程进行说明）
3. 为什么`sleep`（以及`wakeup`）要使用`ptable.lock`？xv6如何通过锁的使用来解决**lost wake-up**问题?
4. `sleep`函数中，2890行if语句的作用是什么?
5. `sleep`函数中，2891行和2892行能不能交换顺序？为什么?
6. 阐述`sleep`函数执行时，进程是如何转入睡眠态，又转入就绪态和运行态，并继续执行sleep的。
7. xv6的`wakeup`操作，为什么要拆分成`wakeup`和`wakeup1`两个函数，请举例说明。
8. 假设`wakeup`操作唤醒了多个等待相同`channel`的进程，此时这多个进程会如何执行？xv6的`wakeup`是否符合**Mesa semantics**?
9. `wakeup`时如果没有`sleeping`的进程，`wakeup`会阻塞吗?

Linux 信号量

请阅读Linux kernel中关于信号量的代码 [include\linux\semaphore.h, kernel\locking\semaphore.c] 以及查找相关资料，回答如下问题：

1. Linux中信号量的数据结构及对应的PV操作；
2. 说明Linux扩展的各类`down`操作的用途；
3. 简要分析`down`和`up`操作的实现。

第三部分

时钟中断

相关文件：

main.c、lapic.c、trap.h、trap.c、x86.h、proc.c、swtch.S、trapasm.S、mmu.h、vectors.pl

重点关注问题：

1. 简述时钟中断的注册过程
2. 时钟中断的处理程序（3414~3422）中，`ticks`变量的作用 （hint：结合`sleep syscall`）tickslock锁的作用是什么?
3. 请详细描述，时钟中断到来之后（从硬件读出`IDTR`寄存器中`idt`的地址开始），内核进程切换的调用流程。（主要包括`wakeup`,`sched`,`yield`,`swtch`,`scheduler`,`trap`,`alltraps`函数）
4. 在上述过程中，为什么scheduler函数的循环体最开始要先开中断`sti()`？明明紧接着的`acquire(&ptable.lock)`立刻就关中断了？
5. `yield()`函数中的`acquire(&ptable.lock)`操作与`release(&ptable.lock)`操作执行过程中，与一般的线程使用自旋锁来保障临界区有什么区别？ (Hint: sleep同理)

wait & exit & kill 系统调用

相关文件

proc.c

重点关注问题：

1. 一个进程从执行`exit`开始，到最终彻底“消亡”（pcb、内核栈、页表等都被回收），进程状态经历了怎样的变化?
2. 一般由父进程A回收子进程B的页表和内核栈，但如果父进程A先于子进程B执行`exit`操作，由谁来回收子进程B的相关数据结构?
3. 请简述xv6是如何利用自旋锁（`spinlock`）和条件变量(`sleep`&`wakeup`)实现`wait`以及`exit`的?
4. xv6中`kill`操作，是如何实现的？进程是在什么时候真正被“杀死”的?
5. 为什么调用`mycpu()`函数的时候（例如：`myproc`（2456）函数）中，需要先关中断？为什么调用`myproc()`函数（例如`wait`、`exit`）的时候不需要关中断?

Linux RCU

RCU (Read-Copy-Update) 是Linux 内核在2.5版本之后提供的一种新的同步机制，请通过查阅资料[Documents/RCU/whatisRCU.txt]，介绍RCU的产生原因、基本原理、核心API，并且任选[Documents/RCU/listRCU.txt]中提供的3个程序示例中的一个进行分析，介绍RCU API的使用。

第四部分

管道

相关文件：

pipe.c

重点关注问题：

1. 管道`pipe`的数据结构是如何表示的？其中重要字段的作用分别是什么？
2. `pipe`中的`nread`以及`nwrite`，在使用过程中，如果超出了缓冲区大小，是否会进行取模回滚的操作？
3. `pipe`中的缓冲区判满和判空条件分别是什么？
4. `piperead`和`pipewrite`函数中，为什么要使用两个条件变量（`p->nread`，`p->nwrite`）？能否只使用一个？如果不能请举例说明。
5. 当只有一个读者和写者的时候，`pipewrite`函数中的6836行以及`piperead`函数中的6856行的`while`语句能否使用`if`代替？多读者多写者呢？

睡眠锁

相关文件：

sleeplock.h、sleeplock.c

重点关注问题：

1. xv6中`sleeplock`的数据结构及其中重要字段的作用
2. `sleeplock`在xv6中的使用场景？
3. 请简述xv6是如何利用自旋锁(`spinlock`)和条件变量(`sleep`&`wakeup`)实现`sleeplock`的？
4. 为什么`sleeplock`在获取到锁之后，直到释放锁的这段时间内，并没有像`spinlock`一样屏蔽中断？

Linux & pthread 读写锁

1. Linux kernel中读写锁主要分为两种类型，分别是：
    1. spinlock类型 [include/linux/rwlock_types.h，arch/arm/include/asm/spinlock_types.h，inlcude/linux/rwlock.h]
    2. 信号量类型 [include/linux/rwsem.h，include/asm-generic/rwsem.h，kernel/locking/rwsem.c]

请阅读相关代码并查找资料，介绍读写锁的由来，特性，基本原理和使用场景，并且选择上述一种类型，给出相应的数据结构和支持的操作。

2. 简要`pthread`提供了读写锁接口，鼓励自行编写一个写的benchmark程序比较`pthread_rwlock`和`pthread_mutex`的性能差异。