操作系统实习报告

曹胜操 - 1500012838

Challenge! Fine-Grained Locking

> JOS中所使用的大内核锁十分方便简单，但是它的缺点在于内核模式下无法实现多核并行。现代的操作系统通常都使用细粒度锁来保护共享状态的不同部分，这可以显著提高系统性能，但是也更难以实现。你可以自己决定锁的粒度，以下划分作为参考：
>
> - 页分配器
> - 控制台驱动
> - 调度器
> - 进程间通信

首先我们需要梳理清楚使用大内核锁的JOS从启动到运行的整个流程：

- `boot/boot.S` `boot/main.c` `kern/entry.S`：加载内核以及最基本的初始化，之后进入`kern/init.c`中的`i386_init()`
- `cons_init()` `mem_init()` `env_init()` `trap_init()`：Lab1至Lab3中的内容所做的一系列初始化
- `mp_init()` `lapic_init()` `pic_init()`：Lab4中针对多核进行的初始化，但此时仍然是只有BSP在运行
- `boot_aps()`：由BSP逐个唤醒APs，在此之前需要`lock_kernel()`
- 对于APs，执行完`mpentry.S`中的代码之后进入`kern/init.c`中的`mp_main()`，同样是在一些初始化之后`lock_kernel()`，由于BSP一定是在此之前获得锁的，APs会在此阻塞直到BSP通过调度进入用户环境释放锁
- BSP创建用户环境，如`ENV_CREATE(TEST, ENV_TYPE_USER)`，然后通过`sched_yield()`调度用户环境
- 如果某个CPU在调度时发现并没有可以运行的用户环境，就会通过`sched_halt()`停机，在正式停机之前还会`unlock_kernel()`
- 当发生系统调用等情况进入trap时，在`trap()`中还会`lock_kernel()`

锁的本意是保护被共享的内容。在JOS中，大多数情况下CPU都是在操作完全属于自己的独立内容，但是这些东西是共享的：

- `envs`：管理用户环境
- `pages`：管理内存页分配
- 外部的I/O端口，主要是控制台：输入输出

于是，我们可以先设计一个较为粗糙的细粒度锁，分别对上述三者进行保护。涉及到上述共享内容的较底层的内核函数主要位于：

- `kern/console.c`：I/O
- `kern/env.c`：主要是`envs`，其中涉及到新建环境时也会分配新的页，影响`pages`
- `kern/pmap.c`：`pages`
- `kern/printf.c`：I/O，比`kern/console.c`高一层
- `kern/sched.c`：`envs`
- `kern/syscall.c`：有一些直接对`envs`的操作
- `kern/trap.c`：有一些直接对`envs`的操作

综合以上的考虑，我们进行这样的设计：

- 在`kern/spinlock.h`和`kern/spinlock.c`中添加四个较细粒度的锁：`ev_lock` `pg_lock` `io_lock` `mo_lock`，取代原来的大内核锁`kernel_lock`
- 在`kern/env.c` `kern/init.c` `kern/monitor.c` `kern/sched.c` `kern/syscall.c` `kern/trap.c`中用锁对上述几个内容进行保护

具体的修改记录如下：

- `kern/env.c`
  - `env_setup_vm` `region_alloc` `env_free`中添加`pg_lock`的保护
  - `env_alloc`中添加`io_lock`的保护
  - `env_destroy()`中如果是当前环境摧毁自己，`env_destroy()`不会正常返回，需要最后释放`ev_lock`
  - `env_run()`的调用者都需要先获得`ev_lock`，在`env_run()`最后会调用`env_pop_tf()`，然后在其中的汇编代码之前才能释放`ev_lock`（释放这个锁的时机调试了很久才得出，如果像大内核锁一样在`env_pop_tf()`之前释放会导致`primes`程序发生一些奇妙的错误）
- `kern/init.c`
  - `i386_init`中启动APs和创建用户环境前后添加`ev_lock`的保护
- `kern/monitor.c`
  - `monitor()`中添加`mo_lock`的保护，保证只有一个CPU运行监视器
- `kern/sched.c`
  - `sched_yield()` `sched_halt()`中添加`ev_lock`的保护，遇到调用`env_run()`的先不释放`ev_lock`
- `kern/syscall.c`
  - `sys_cputs()`中添加`ev_lock` `io_lock`的保护
  - `sys_cgetc()`中添加`io_lock`的保护
  - `sys_env_destroy()`中添加`ev_lock` `io_lock`的保护
  - `sys_exofork()` `sys_env_set_status()` `sys_env_set_pgfault_upcall()`中添加`ev_lock`的保护
  - `sys_page_alloc()` `sys_page_map()` `sys_page_unmap()`中添加`ev_lock` `io_lock`的保护
  - `sys_ipc_try_send()` `sys_ipc_recv()`中添加`ev_lock` `io_lock`的保护
- `kern/trap.c`
  - `print_trapframe()`中添加`io_lock`的保护
  - `trap_dispatch()` `trap()` `page_fault_handler()`中添加`ev_lock`的保护，遇到调用`env_run()`的先不释放`ev_lock`

修改完成后`make grade`通过，看起来应该是没有问题了。