操作系统研讨课 实验报告

代瀚堃 2019K8009929051

一、实验中遇到的问题

1. 打完补丁后还需要手动 Merge 代码，太费时间

是有点费时间，需要耐心

2. 新增的进程管理函数一次性写完，各种 bug 混杂在一起

没有遵循增量式开发的原则，由于新增的几个系统调用都有一定的相似性，于是我将它们一并写好，才到 QEMU 上运行，果不其然是有 bug 的，而且由于很多 bug 叠加，加上我在添加这些方法的时候，还对原来调度器的逻辑进行了调整，且没有经过测试，使整个内核呈现出及其复杂的行为，很难理清逻辑，最后无从下手，只好从 Project 2 的代码开始重构，改好一个模块，测试一个模块。其中就遇到了 FAQ 中提到的玄学 bug——enable_preempt 的问题，原因是我改动了 PCB 的结构，在栈顶寄存器的后面又加了两个成员变量，内核和用户的栈底，但头文件 regs.h 中 preempt_count 变量位于内核 sp 和用户 sp 之后，导致 printk 在调用 enable_preempt 和 disable_preempt 时使用了 sp 的低几位，将中断关闭。这个 bug 难调之处在于，我们启动 QEMU 并初始化屏幕之后，便没有任何变化，只知道时钟中断被莫名其妙地关掉了，但整个过程中我找不到是在哪里关的，因为我在 set_sbi_timer 并回到用户态后，并没有调用过 disable_interrupt 这样的函数，最后只好用 gdb 一步一步跟着调试，才发现在 enable_preempt 中取出的 preempt_count 值是栈指针的值。

在这些进程管理函数调试的最后阶段遇到的 bug 是调用 sys_kill()函数后，内核崩溃，或者是能够杀死一个进程，但是当杀掉第二个进程时会崩溃，原因是我在让一个进程正式退出后（完全退出，僵尸进程不算是完全退出），会把它的 node 域挂到 exit_queue 上，并恢复栈指针，但后面的进程再复用已经退出的 PCB 时，并没有把它从 exit_queue 中取回来，导致进入到该进程时跑飞了。

3. 将邮箱发送/接收封装成系统调用，发送/接收失败后无法恢复

mbox_recv/mbox_send 封装成系统调用，当发生阻塞时直接将进程挂起到 mailbox 的阻塞队列中，等到释放时没有恢复消息的传递，最后只好使用同步原语，在获取资源失败时不断尝试重新传输，类似于自旋的效果。

以上 bug 基本上每个都花费了 2 个小时的时间，一个很小的 bug 就有可能让整个系统呈现出复杂的行为，让人难以下手，只能先粗粒度地确定出在哪一范围发生了错误，再细粒度确定到某一函数，某一语句，甚至于是一条汇编指令。调试起来十分花时间，虽然想怎么修改倒是不难，可以说找到了 bug 就能想到解决方案。

4. 双核的调试

让两个 CPU 核都正常工作可以称作是“驯服”，因为两个 CPU 是并行工作的，刚开始很难控制它们的行为，而且调试起来只能看到其中一个核，比较困难。我遇到的第一个比较难调试的 bug 是没有将两个核

的栈指针分开，导致从核被唤醒后干扰到了主核的运行，这个 bug 花了我很长时间的原因是我只看了 bootblock 和 main 函数，而忽略了中间的 head.S 部分，sp 指针恰恰就是在这部分设置的，我在进入 main 函数后显示了 sp 寄存器的值才发现问题。

还有一个很难调试的 bug 是我在初始化 PCB 栈时写到了非法区域，这个 bug 离奇的地方在于，我在 QEMU 上运行不会出错，只是初始化屏幕后没有反应，而用 gdb 调试时，调用 init_pcb_stack 有时候会出错，有时候又不会出错，我一直以为应该是调度器的问题，没想到前面的 init_pcb_stack 就已经出问题了，这个 bug 表现为，我用 kmalloc 分配一段空间时，右边的变量值是正确的，但赋到左边就变成了 0，所以向低地址写入数据，硬件报错。原因未知，受 Project4 启发，我猜测是内核扇区数目太多（100 个扇区），SD 卡没有按照预期将它们一次性读入，导致有部分数据缺失。后来改动了 bootblock 就能正常运行。

二、还有待解决的问题

1. 目前我的信箱仅支持多输入单输出，而不支持多输入多输出，一旦启动多个接收端，就会触发异常
2. 信箱的操作相当于是用一把大锁（值为 1 的信号量）实现的，效率不高，阻塞次数较多
3. Project1 中我们的 bootloader 支持内核重定位到 0x50200000 处，但目前这个实验似乎没有好的解决方案，如果将原来那段代码覆盖掉，从核可能没有办法运行