操作系统 Lab 2

151242041, 王昊庭, hatsuyukiw@gmail.com

2017 年 4 月 9 日

1 实验内容

完成了从内核态和用户态之间的切换。具体地说，现在游戏与内核已经分开编译。游戏对于串口（终端）和屏幕的操作均使用第 0x80 号中断，即系统调用实现。

助教先生（或者女士）可以使用这样的方法简单验证我的游戏运行于用户态上：`game` 目录下的 `game_test.c` 使用了特权指令,更改 `game.ld` 可以将入口函数改为 `void game_test()`，如此再运行游戏会发生第 13 号异常，即通用保护错。

我才用了扁平模式 + 分页的方式管理内存，查看 `kernel/kernel.ld` 可以验证 OS 运行在高端地址上。

由于时间有限，并且目前只运行一个进程，负责进程的资源（包括段和页等等）的回收的代码还是半成品，未测试，所以根据 JYY 机器第二公理，都是错的。请不要运行它们。

2 实验中遇到的问题

本次实验中最难调试且最意想不到的一个 bug 是，OS 已经开启了分页，但是却无法对某一个“有效”的虚拟地址进行操作，具体地说是发生了如下的情况，

```
*(int *)(0x08000000) = 100; // No Error
printk("%d\n", *(int *)(0x08000000)); // 0
```

检查 CR3 寄存器、页目录和页表均没有问题。

最后发现是因为 QEMU 默认只提供 128MB 内存，而 `0x08000000` 的虚拟地址被映射到了 `0x10000000` 的物理地址。总之我觉得在执行第一条语句，即赋值的时候不报错是一个很费解的行为。

我没有 Linux 的系统进行测试，如果存在兼容性问题请务必联系我，谢谢。

3 实验中的一些 trick

我认为 JOS 手工填写页目录和页表的方法中有一个很有趣的细节。`kernel/entry.S` 的注释提出了一个问题，即在开启页表的瞬间之后，内核代码对于 CPU 来说就已经存在在高端地址了，但是 EIP 寄存器中的值仍然是低端地址，如何保证 CPU 继续按顺序执行接下来的代码呢？答案是内核做了两次内存映射，一个页目录将高端地址映射到物理地址，另

一个将与物理地址相同的虚拟地址映射到物理地址（单位映射）。这样可以保证开启分页前后无缝衔接。

我观察到低端地址事实上并没有被使用，因此我在开启完整页表后仍然保留了这个性质，即我将 `0x10000000` 以下的虚拟地址做了一个单位映射，这样我之前在内核中写的操作物理地址的代码就不用更改了。

这个偷懒行为应该是不太可取的，因为这样的话更改一个虚拟地址的值会影响另一个虚拟地址的值，这对机器来说是费解的。不过既然是内核，一切都在程序员（也就是我）掌握之中，那就无所谓了吧。

一个完整的页目录和页表需要 4MB 多一点的空间，而 JOS 的手工填写页目录和页表的方法只映射了一个页目录，即 4MB 的空间。因此没有办法在 4MB 的空间内开启完整分页模式。可以是先填写只映射 `0xf0000000` 上方的高端地址的页表，之后再为完整的页表分配空间。这样的话内核总共使用了三个不同的页目录和页表，真好玩！(虽然我并没有这么做。)