oslab2 实验报告

姚荣春 151110079

1. gcc 版本及联系方式

Ubuntu 4.9.3

15895873878@163.com

2. 问题回答

1）建议你搭上顺风车，让内核就跑在从 0 开始的物理地址中，至于为什么，你可以思考如果不这么干应该怎么做。

答： 如果不让内核跑在从 0 开始的物理地址中，可以把链接文件把 kernel 的起始地址改为其它的地址。或者在 load 的时候手动加一个偏移量，然后在段描述符中补上那个偏移量即可。

2）为什么我们需要考虑这些？很重要的原因是： 系统调用过程中如果涉及到指针的传送，由于段式存储的特点，我们需要知道指针对应的段到底是哪一个，这样我们才能获取对应的基地址。 到了这里，请你仔细思考为什么扁平模式下我们不需要考虑这个问题。

答： 因为扁平模式下的基地址都是 0，因此使用任意的 ds 段描述符都是一样的。我们不需要特权级也能获取到内核（任何程序，因为都是 0）的基地址。

3）经过讨论，我们发现还是将代码段，数据段，栈都放到一个段里比较好，虽然这样会比较危险（用户栈向下增长时会将我们的代码和数据冲掉），但这是目前最实际的实现方法了。在这里请思考段页式存储是如何解决这个问题的。

答：　　段页式储存可以把段设置成扁平的，加载代码数据时把内存使用状况存下来。而页表通过储存好的“已使用”信息来把虚拟地址指向没有用过的地址，不妨这样：在页故障的时候分配一个空闲的空间。这样用户栈就不会冲掉代码和数据了。

4）陷阱帧初始化特别注意事项：eip，中断返回现场，请你自行思考应该设置为哪个值

答：　　设置成游戏的入口地址。

5）也许你该仔细地思考为什么每个进程都需要自己的内核栈。

答：　　防止不同的进程相互影响，便于调度。否则比如，一个进程等待 IO，我们可以运行别的进程，如果新的进程也进入了内核栈，储存了局部变量，就会污染之前的内核栈，之前进程 IO 结束后无法很方便的调度。如果都有自己的内核栈就没有这个问题。

3.实验心得

1）在 lgdt 的时候结构体的对齐出了问题：。改成 3 个 16 位的数据就成功了。

2）把图像的系统实现写成刷屏比写成刷点要快很多。这是因为不需要频繁的陷入内核。

3）完成系统调用 sys_call 的 trap_gate 的时候，不小心把其特权要求设立成了系统的特权级，最后才发现，这导致我 qemu 进入用户态之后崩溃了很久。

4）设立陷阱帧的时候，先给 esp 赋予了陷阱帧的首地址，但是忘记了我的陷阱帧结构第一个数据并不是 eip，最后补上了偏移量，解决了这个问题。

5）实现用户的 print 函数的时候，只把第一个参数传给了内核，但是这样会导致内核 printk 接收的第一个参数的地址和用户的 print 的第一个参数地址不同，从而使得内核无法通过偏移来得到其它参数。最后解决方法是：print 把字符处理成最终字符串之后，通过系统调用给内核直接输出，这样内核只需要得到一个参数即可。

6）忘记给用户程序的 EFLAGS 打开 IF 位。

7）进入用户程序之前，忘记给 ds 用户权限，给的是内核权限，这样会导致 iret 的时候把 ds 清空。最后在进入用户程序前赋予 ds 的 ring3 权限解决了这个问题。

8）做这个实验的时候助教不厌其烦的给了我很大的指导。非常感谢。心得：关键时刻要抱助教大腿。

4.makefile 使用介绍

make run 编译运行

make clean 清除.o 等文件

make run 之后运行 terminal.sh(需要加权限)连接 qemu