OS Lab1 文档

Introduction：

介绍了这门课lab需要配置的环境以及讲解了一些git版本控制的简单用法。

我采用了给的虚拟机+apt get except来做该lab。

Part1:

主要目的是让我们了解从开机到kernel启动的一系列过程。查看了bios执行的汇编代码和位置，通过源代码和gdb调试复习了一遍上课学习的第一个sector中引导代码。

同时，学习了qemu的使用方法。在刚刚boot的时候，gdb仅有很少的指令可以生效，si执行下一条指令，x/I addr查看addr的内容。

值得注意的是，不要将物理内存的构造和虚存的堆栈结构搞混。

Part2:

当mbr执行完毕后，kernel被加载到0x100000后，控制权交给了kernel。此时，link address和物理地址的映射已经打开。

这个部分还帮助我们复习了一下elf和调用栈的结构。

Part3:

这个部分开始，开始coding了，主要是补全jos中的一些函数，和增加kernel可以调用的command。讲一下几个部分的实现方法吧。

1. 实现八进制

这个比较简单，按照旁边的十六进制、十进制一样的方法即可。其实是做了提取数字、设置进制然后交给printnum来输出即可。值得注意的是八进制需要在最前面putch一个0

1. 实现强制+

采用的方法是增加了设了一个posflag来表示打开了强制+模式。

然后在format 有符号十进制数（因为这个format里面只有十进制数有符号数）的时候，如果posflag打开了，就多putch一个+即可。

1. 实现右对齐

这个部分实现起来踩了一些坑，一开始没有理解printnum的真正意思，简单的以为只要把左对齐的print反一下就可以。后来通过测试发现有问题，然后仔细去读了printnum这个函数。才发现原来是一个从左print到右的递归。通过递归底层的else打印前缀，然后从高位到低位打印数字。

于是采取了拆分这个函数的方法，判断如果是左对齐，安原有的逻辑执行。如果是右对齐，先递归打印数字，然后再打印右边的补全空格。

1. 实现backtrace

这个部分ics和编译原理都学过很多次了。原理就是通过ebp保存的链表不断找到上层的caller，然后再通过固定的堆栈结构来调整指针获取到eip和args，这里指针写的有点晕，来回调试了好几次。第二部分获取函数信息和行数信息实现比较容易，按照注释和前后几个类似操作调用binarysearch获取信息即可。不过原理需要理解，stab中记录了调试时候的信息，通过search可以去获取这些信息的地址，取到它们。

1. 实现timer

readebp同一个文件里有个执行汇编命令rdtsc的已经封装好了的函数。在开始执行命令时获取一下cycle，结束后再获取一下，两者相见就能得到执行这个命令的cycle数，模拟一个timer的效果。这里对我的难点是argc相关的操作很不熟练，以前很少操作命令行参数，所以花了一点时间看了一下这部分。