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
2. 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。
3. 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。
4. 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。

5. 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。