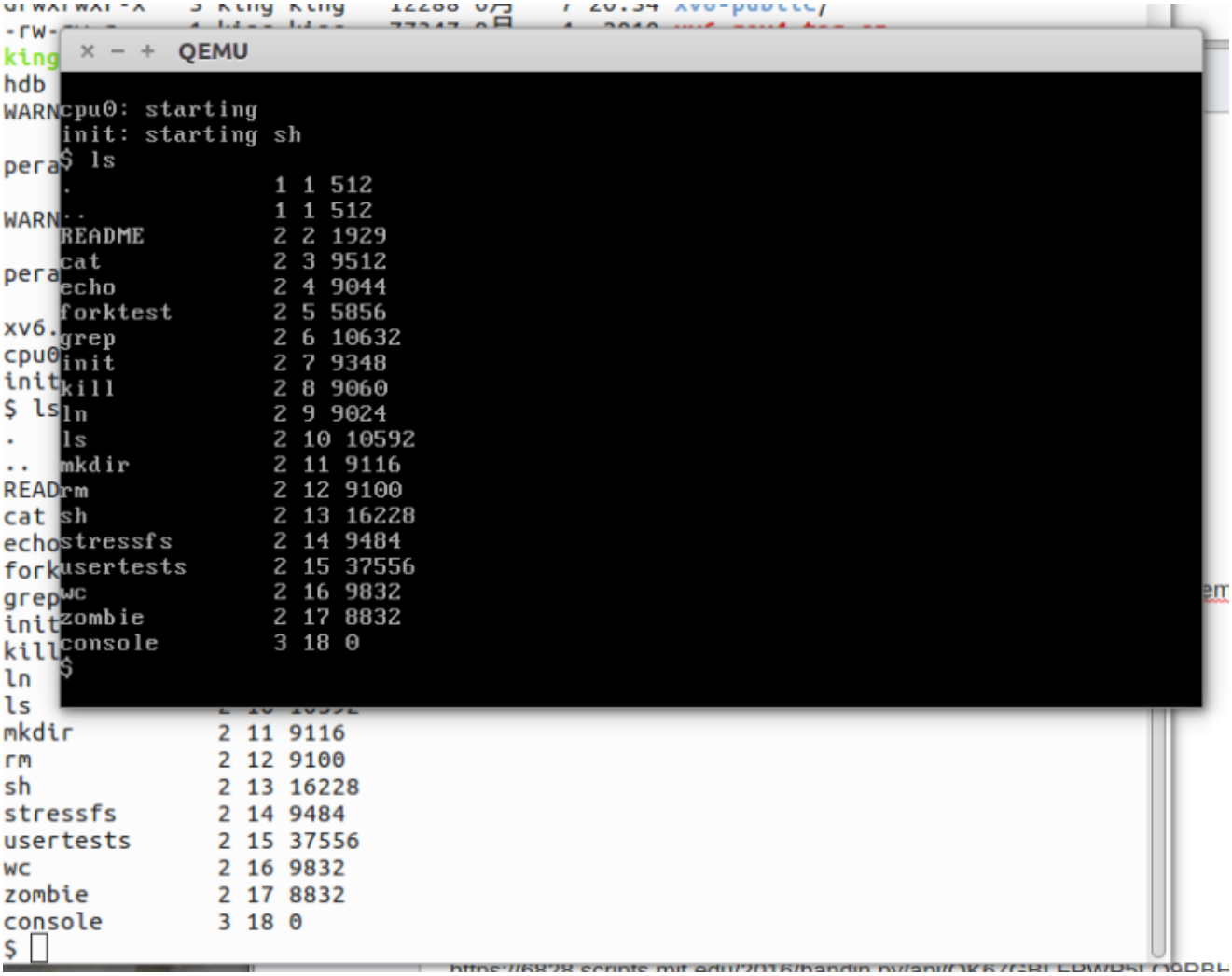
xv6及Labs课程项目



xv6 和 jos 关系：

多核操作系统课程使用的是MIT的JOS操作系统，但是MIT的6.828:Operating System Engineering 课程官网上着重介绍的却是xv6。浏览了下官网，了解了一下这门课程的历史，同时对MIT的大牛老师和大牛同学们满怀敬意。

很长一段时间MIT并没有开设操作系统课程。MIT2002年秋天，MIT的几位老师决定开一个实验性课程（编号6.097）教授操作系统。在教学过程中，同学们依据第六版Unix和John Lions的著名的代码注释逐步写出了一个微内核（exokernel）的操作系统，命名为JOS。从2003年开始，操作系统课程成为了MIT的正式课，编号6.828。

但是第六版Unix使用的是非ANSI C，而且是基于PDP-11架构，不支持intel x86架构。于是MIT的老师和同学们与时俱进地用ANSI C在x86上实现了Unix v6的基本功能,并且实现对多核的支持，这就是xv6操作系统了。xv6还改善了Unix v6中几个较粗糙的地方，比如文件系统和调度算法。

官网没提到是否对2002年的JOS进行了改写，但根据我们的实践，JOS已经是用ANSI C进行了改写，但是和xv6相比还是有若干局限性。比如不支持多核，只有一个内核堆栈（xv6中每个进程有自己单独的内核堆栈）等。

我们的课程名字叫做多核操作系统，似乎用xv6才更符合多核要求。不过这只是为了应付intel的赞助……我们的水平还是只能用JOS，而且JOS都学得够呛。实际上，官网也提到了，Courses taught at UCLA, NYU, Peking University, Stanford, Tsinghua, and University Texas (Austin) have used Jos without xv6。

简单总结来说：

- xv6 是一个完整的，运行在多核环境下的简单操作系统
- jos 是为了完成作业的一个简易操作系统，运行在单核上，内存管理也更简单。

作业完成方式

作业有两种选择方式：

1. 选择读**xv6** 源码
2. 选择做 **Lab**

这两个选择都十分困难，这里不给出建议选择哪一种，可以事先大概了解再做决定。

阅读源码

repo

下载源码并编译运行：

https://github.com/mit-pdos/xv6-public.git

for v6 and plus

```
sudo apt install -y build-essential gdb
// 等等一系列流程参照 Labs 的 Tools章节 安装好 qemu emulator
// 最后对于 xv6-public 项目会编译出 xv6.img 和 fs.img
// 在 Linux 环境安装好 qemu 虚拟机后可以在命令行执行
qemu-system-i386 -serial mon:stdio -hdb fs.img xv6.img -smp 1 -m 512
// 或者执行
qemu-system-x86_64 -smp 1 -parallel stdio -hdb fs.img xv6.img -m 512
// 就可以运行起qemu
```

注：事实上运行起**xv6**并没有什么用。除非你修改**xv6**的 **makefile** 让它支持使用**gdb**调试。

要求

- 引导流程，gdt初始，实模式跳转保护模式过程(涉及汇编，bootasm.S bootmain.c asm.h mmu.h)，一直到进入main函数的整个过程(比较难)
- 多核启动(不需要掌握，但是在阅读过程特别是整个初始化中必不可少)
- 内存管理模型 kinit() kvmalloc(), exec(), allocuvm()
- 内存管理函数，涉及进程页表切换 switchkvm(), 新建进程的内存申请和结束进程的内存释放
- 自旋锁的运作->真正理解死锁原理
- 进程的创建过程与文件加载并执行
- 异常，中断调用流程，系统调用过程(syscall)
- xv6进程切换-swtch函数
- 进程调度，重点是调度器scheduler()
- Shell 的整体代码及调用。
- 管道实现
- 文件系统(不用管日志恢复那些)

对于这12个(11个)要求，

1. 任意选择其中 **5 - 6** 点个做深入了解(注意选择最好主要集中相关联的部分) (可以少于5个，但是不能太少)
2. 对每个相关的模块仔细阅读源码并搞懂运行流程3
3. 对相关的源码做出注释
4. 写出每个点的总结报告

最后提交：

- 加过注释的源码
- 总结报告

注：不要抄博客的内容，要有自己的理解

建议

直接只阅读源码是绝对读不懂的，这是忠告。一个操作系统底层的实现设计大量的底层的前置知识，包括汇编，CPU指令，CPU寄存器，中断描述符，GDT，内存分布等等一系列的知识。这些知识若遇到不懂的，可以先跳出XV6的框，先去大概搞懂是什么，再尝试结合xv6相关的

参考

xv6中文文档 https://th0ar.gitbooks.io/xv6-chinese/content/content/preface.html

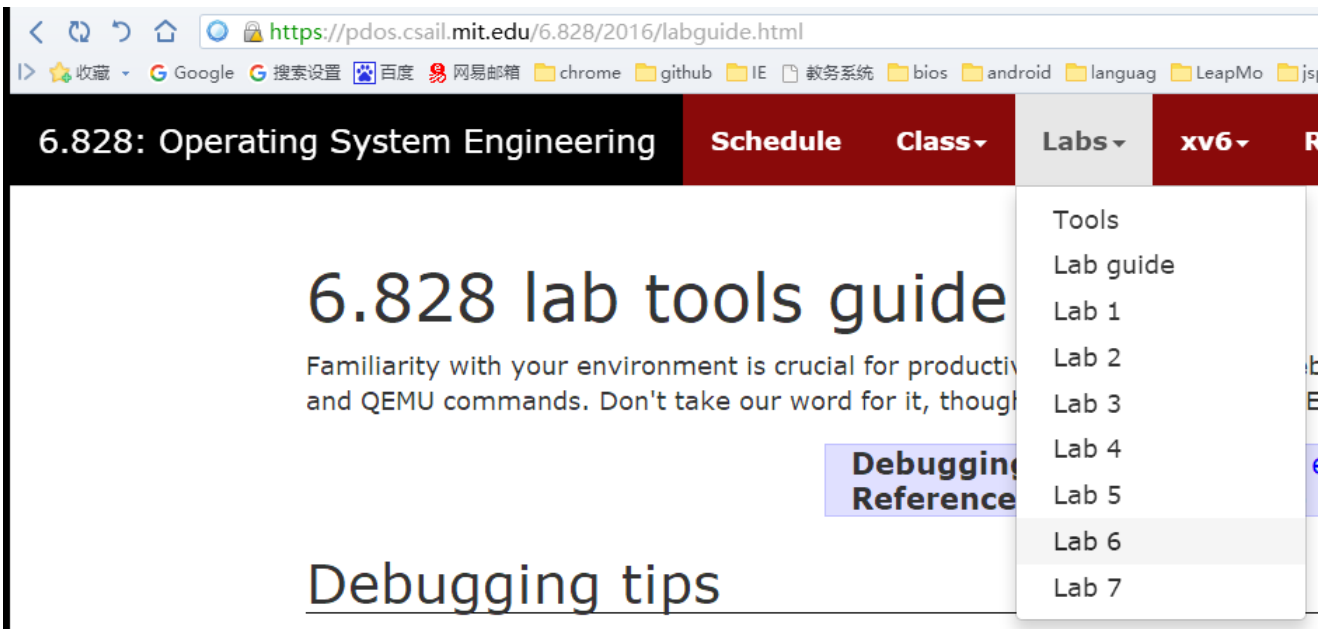
xv6 中文注释源码及 入门部分博客详解 http://leenjewel.github.io/blog/categories/cao-zuo-xi-tong/

xv6 源码详解博客 http://blog.csdn.net/qq_25426415/article/category/6684908

xv6 其他辅助资料 http://blog.csdn.net/Swartz2015/article/category/6071599

Labs => JOS

Labs官网 https://pdos.csail.mit.edu/6.828/2016/labguide.html



学习顺序

Tools->Lab guide-> (Lab 1) ->Lab 2-> **Lab 3**->**Lab 4**->**Lab 5**

要求

至少完成

- Lab3
- Lab4
- Lab5

虽然只说完成这3个，但是实际上要完成这3个Lab需要很好的搞懂Lab1(目的是为了了解好环境搭配与调试环境)

Lab2作为其中的过度也是不可缺少的，不过因为其太偏向底层的内存管理，实质上不用了解的特别清楚，但是还是需要知道大概的流程，所以不用完成的很完整。

Lab3是工作量最大的，但是也是收获最大的一个Lab

把Labs做完，最后**提交**

- 保留代码
- 完成**Labs**过程中的文档。**(**可以理解为答题过程的那种**)**

简单开始运行**Labs**初始

QEMU is a modern and fast PC emulator. QEMU version 2.3.0 is set up on Athena for x86 machines in the 6.828 locker (`add -f 6.828`)

Unfortunately, QEMU's debugging facilities, while powerful, are somewhat immature, so we highly recommend you use our patched version of QEMU instead of the stock version that may come with your distribution. The version installed on Athena is already patched. To build your own patched version of QEMU:

1. Clone the IAP 6.828 QEMU git repository `git clone http://web.mit.edu/ccutler/www/qemu.git -b 6.828-2.3.0`
2. On Linux, you may need to install several libraries. We have successfully built 6.828 QEMU on Debian/Ubuntu 16.04 after installing the following packages: libsdl1.2-dev, libtool-bin, libglib2.0-dev, libz-dev, and libpixman-1-dev.
3. Configure the source code (optional arguments are shown in square brackets; replace PFX with a path of your choice)
    1. Linux: `./configure --disable-kvm [--prefix=PFX] [--target-list="i386-softmmu x86_64-softmmu"]`
    2. OS X: `./configure --disable-kvm --disable-sdl [--prefix=PFX] [--target-list="i386-softmmu x86_64-softmmu"]` The `prefix` argument specifies where to install QEMU; without it QEMU will install to `/usr/local` by default. The `target-list` argument simply slims down the architectures QEMU will build support for.
4. Run `make && make install`

注：如果不打算把qemu安装在`/usr/local` 目录下，那么要把--prefix=/xxx设置好

然后把 `/xxx` 添加到path变量中`export PATH=$PATH:/xxx`

lab1:Booting a PC

```
athena% mkdir ~/6.828
athena% cd ~/6.828
athena% add git
athena% git clone https://pdos.csail.mit.edu/6.828/2014/jos.git lab
Cloning into lab...
athena% cd lab
athena%
```

hand in 部分不用管

首先先编译源码

```
athena% cd lab
athena% make
然后运行：
athena% make qemu
```

之后会显示

```
K> help
help - display this list of commands
kerninfo - display information about the kernel
K> kerninfo
Special kernel symbols:
  entry  f010000c (virt)  0010000c (phys)
  etext  f0101a75 (virt)  00101a75 (phys)
  edata  f0112300 (virt)  00112300 (phys)
  end    f0112960 (virt)  00112960 (phys)
Kernel executable memory footprint: 75KB
K>
```

此时就是正式运行

调试方法：

退出刚才的模拟器

打开两个终端

其中一个终端在`lab`目录下运行

```
make qemu-gdb
```

此时模拟器会卡住

在另一个终端里面运行

```
gdb
>Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.xxxxx
....
```

之后再gdb的这个终端里面输入

```
target remote localhost:26000
symbol-file obj/kern/kernel
```

然后按 `c` 就可以发现原来的卡住的模拟器开始执行了。

其他的方式就需要熟练使用gdb之后才能灵活操作了。

若不熟练使用gdb也可以做，只是就会觉得很不方便。

参考

xv6中文文档 https://th0ar.gitbooks.io/xv6-chinese/content/content/preface.html

xv6 中文注释源码及 入门部分博客详解 http://leenjewel.github.io/blog/categories/cao-zuo-xi-tong/

xv6 学习笔记 http://www.cnblogs.com/fatsheep9146/category/769143.html

xv6 另一个学习笔记 http://blog.csdn.net/woxiaohahaa/article/category/5886189