一、实验目的

- 搭建操作系统实验环境，掌握编写操作系统的基本方法
- 接管裸机的控制权
- 练习nasm语言的编程

二、实验要求

- 虚拟机安装，生成一个基本配置的虚拟机XXXPC和多个1.44MB容量的虚拟软盘，将其中一个虚拟软盘用DOS格式化为DOS引导盘，用WinHex工具将其中一个虚拟软盘的首扇区填满你的个人信息。
- 设计IBM_PC的一个引导扇区程序，程序功能是：用字符'A'从屏幕左边某行位置45度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加你的个性扩展，如同时控制两个运动的轨迹，或炫酷动态变色，个性画面，如此等等，自由不限。还要在屏幕某个区域特别的方式显示你的学号姓名等个人信息。将这个程序的机器码放进放进第三张虚拟软盘的首扇区，并用此软盘引导你的XXXPC，直到成功。

三、实验方案

（一）、基础原理

1、操作系统接管裸机的原理：

在没有操作系统的计算机上，仅有BIOS提供基本的中断服务。BIOS会在可能的引导介质上寻找操作系统引导程序。如果引导介质的第一个扇区的最后两个字节为55AA，那么这个扇区将被当做有效的主引导记录。改扇区的内容从第一字节开始被当做代码区加载入内存0000:7c00处，裸机控制权从此刻交给操作系统。

2、在屏幕显示字符的原理：

显卡作为一个外设，CPU指令能够操作的是它的缓冲区（显存）。通过操作显存，可以将字符显示在显示器上。通过一定的程序逻辑，就可以实现要求的弹射效果。

屏幕缓冲区的大小为25行80列，显存的起始地址是0xB800h，显存中每两个字节表示一个字符，高字节决定字符前景色，背景色，是否闪烁；低字节是字符的ASCII码。

除了直接修改显卡缓冲区外，还可以调用BIOS10号中断，其中提供了清屏、显示字符串等功能。

（二）、实验工具和环境

1、实验环境：

由于我的电脑是MacBook，出于方便考虑实验在Mac OS上进行。Mac OS是Unix兼容系统，实验所用到的大部分工具均为类Unix（包括Linux）系统上通用的工具软件。因此本文所报告所描述的试验方法很容易在更为广泛使用的Linux上复现。这些工具也有Windows版本，可以通过Windows Linux Subsystem或Cygwin等方式获取。

2、实验工具

编写操作系统的基本流程是：

这一过程需要一个完整的工具链支撑：

(1) 代码编辑器vim

vim是类Unix操作系统内建的文本编辑器，因其独特的“命令式”编辑方式因此编辑代码十分高效。

(2)软盘创建工具bximage

bximage是一个开源的交互式软盘创建工具。可以创建空白的软盘镜像。

(3)汇编语言编译器nasm

本次实验中仅用到了汇编语言编程。使用到了nasm。Nasm可以生成BIN/ELF等多种格式的机器代码。

(4) 磁盘修改工具dd

dd是单一UNIX规范标准所规定的标准命令，主要功能为转换和复制文件。可以将二进制代码写入软盘镜像。实际上也具有创建软盘镜像的功能。

(5) 十六进制编辑器

对于镜像可能需要手动进行编辑。这里我使用了Mac专有软件Hex Friend，在Linux系统上可以用ghex2替代，在Windows系统上可以用Winhex替代。

(6)虚拟机软件VMware/qemu/bochs

VMware是大型的商业虚拟机软件，有容易使用的图形界面，在Mac下的版本为VMWare Fusion。

qemu/bochs则是两款开源的虚拟机软件，在本次实验中我均有尝试。

相对于VMware，qemu/bochs有以下优势：

- 启动迅速，几乎是瞬间启动
- 只需要简单提供命令行参数或给定启动脚本就可以启动，不需要麻烦的创建虚拟机过程
- 方便调试，这一点对编程十分重要

因此本次实验使用了VMware建立Dos虚拟机（方便切换软盘），使用qemu/bochs运行自己写的引导扇区程序。

qemu和bochs两个使用起来的主要区别是qemu功能更为强大，速度更快，但需要和gdb通过tcp连接进行调试。bochs则有内建的调试控制台。

(7)调试器gdb

刚刚提到qemu需要连接gdb进行调试。

(8)自动化构建工具make

编写完代码后，从编译到写入镜像到在虚拟机执行的整个过程基本是固定的，我编写了Makefile，使用自动化的方式节省时间。

我在本实验中主要用到以下两条规则

- 执行make qemu即可完成整个编译、写软盘镜像、调用qemu在虚拟机中执行的过程
- 执行make bochs即可完成整个编译、写软盘镜像、调用bochs在虚拟机中执行的过程

如果中间步骤出错，make将会终止并提示错误信息。与这两条规则有关的Makefile如下（整个Makefile较长这里不贴出来了）：

```
#目标文件名,请按照实际的汇编文件名修改
OBJFILE = lab1

#软盘文件名
FLOPPYFILE = floppy.img

#编译规则
obj: $(OBJFILE).asm
    nasm $(OBJFILE).asm -o $(OBJFILE)

#生成软盘规则
floppy: obj
    dd if=/dev/zero of=$(FLOPPYFILE) bs=1024 count=1440
    dd if=$(OBJFILE) of=$(FLOPPYFILE)  conv=notrunc

#指定qemu程序
QEMU = qemu-system-i386
```

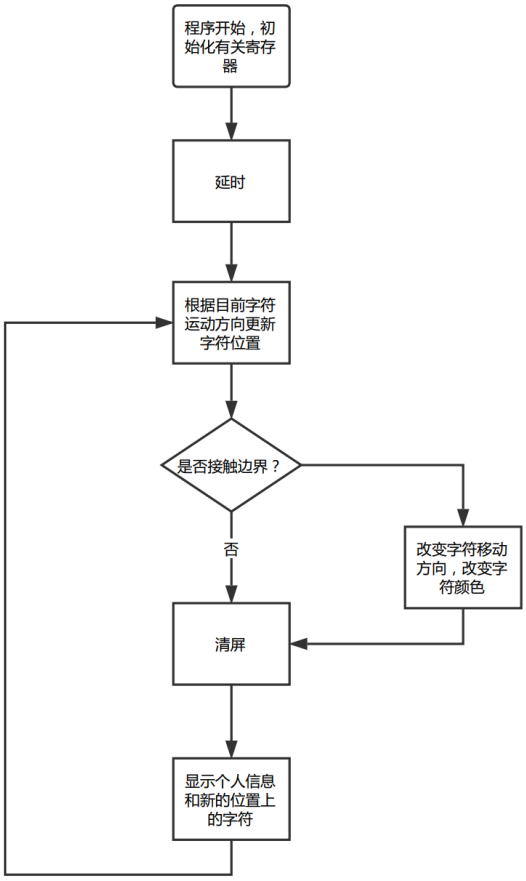
```
#qemu执行参数
QEMUOPTS = -drive file=floppy.img,index=0,format=raw -smp 1 -m 4
$(QEMUEXTRA)

#执行qemu虚拟机
qemu: floppy
    $(QEMU) -serial mon:stdio $(QEMUOPTS)

#启动bochs虚拟机
bochs: $(FLOPPYFILE)
    export FLOPPYFILE=$(FLOPPYFILE) && bochs -f bochsrc

#清理
clean:
    rm $(OBJFILE) $(FLOPPYFILE)
```

(9)版本控制软件git

由于本操作系统是采用原型进化的方式开发，有时想到一个新的功能可能会去尝试，但如果不能实现，就要退回到以前能够正常工作的版本。对于一个功能有时也有不同的实现思路想去尝试，就要同时存在多个分支。因此就要采用一个版本控制工具。这里采用的是git。

（三）、程序流程和算法思想

实验的任务三（设计一个引导扇区程序，实现发射一个字符，不断反弹，并显示个人信息）是本实验中最有难度的部分。程序的流程图如下：



在实验的基本要求之上，我还实现了以下功能：

- 每次字符触碰到边缘时改变字符颜色。
- 每次移动字符时清除之前的字符。
- 保证个人信息不被覆盖。

（四）、数据结构与程序模块功能说明

(1)字符运动的控制

本程序采用两个变量x,y记录字符当前位置。任何时刻x,y方向的运动速度大小的绝对值均为1，而运动方向分为右下，右上，左下，左上四种情况，下一时刻坐标加上速度的值，得到新的位置。

边缘触碰的检测是根据运动方向当前字符坐标减去边缘的一个坐标值，如果为0说明触碰到了边缘，速度将会反向。

(2)模块化功能

代码的组织采用了模块化思想，将4个被多次调用的功能（清屏、显示个人信息、改变字符颜色、延时）封装为类似C语言中的函数：

- 清屏

```
cls:
    pusha           ;保存寄存器的值
    mov ah,0x06     ;调用10号BIOS中断的6号功能
    mov al,0        ;al=0代表清屏
    mov bh,0x07     ;设置将屏幕置为黑底白字
    mov ch,0        ;从(0,0)到(24,79)
    mov cl,0
    mov dh,24
    mov dl,79
    int 0x10        ;调用中断
    popa            ;恢复寄存器的值
    ret             ;返回
```

- 显示个人信息

```
print_id:
    pusha
    mov ax, myid
    mov bp, ax      ;es:bp: 字符串首地址
    mov cx, 17      ;字符串长度
    mov ax, 01300h  ;调用Write string功能
    mov bx, 00F1h   ;白底蓝字，闪烁
    mov dx, 00920h  ;显示在屏幕中央
    int 10h
    popa
    ret
```

- 改变字符颜色

```
change_color:
    cmp byte[color],0Fh ;当前字符颜色是否为最后一种
    jnz no_rst          ;如果不是，选择下一种
    mov byte[color],0   ;如果是，重置
no_rst:
    inc byte[color]     ;选择下一种
    ret
```

- 延时

```
;由于单个寄存器的最大储存的值有限
;故采用两个存储器，两层循环延时
sleep:
    pusha
    mov cx, ddelay
    OUTER:
        mov bx, delay
        INNER:
            dec bx
            jg INNER
    loop OUTER
    popa
    ret
```

(3) 使用代码填充扇区内容

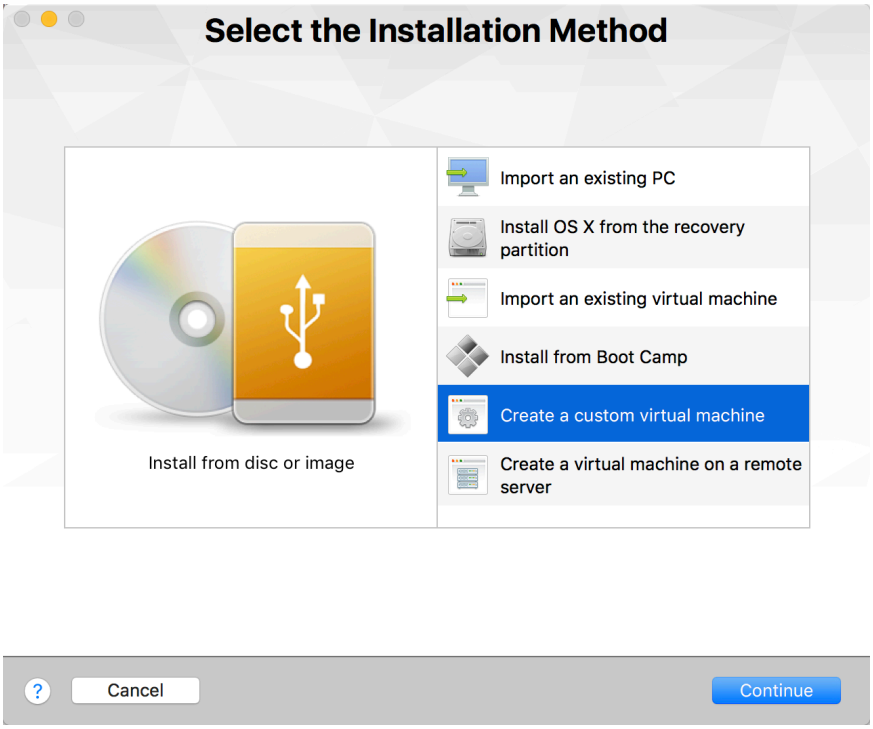
在程序的结尾处使用了times命令填充了程序不足512个字节的部分，并写入了55AA主引导扇区标志。这一操作使得不必每次编译后都使用十六进制编辑器手动填充扇区和填写标志。

```
;$指代码当前行，$$指代码当前section的开始地址
;本程序仅一个section，故($-$$)指当前行离开头的距离
times 510 - ($ -$$)     db  0
dw    0xaa55
```

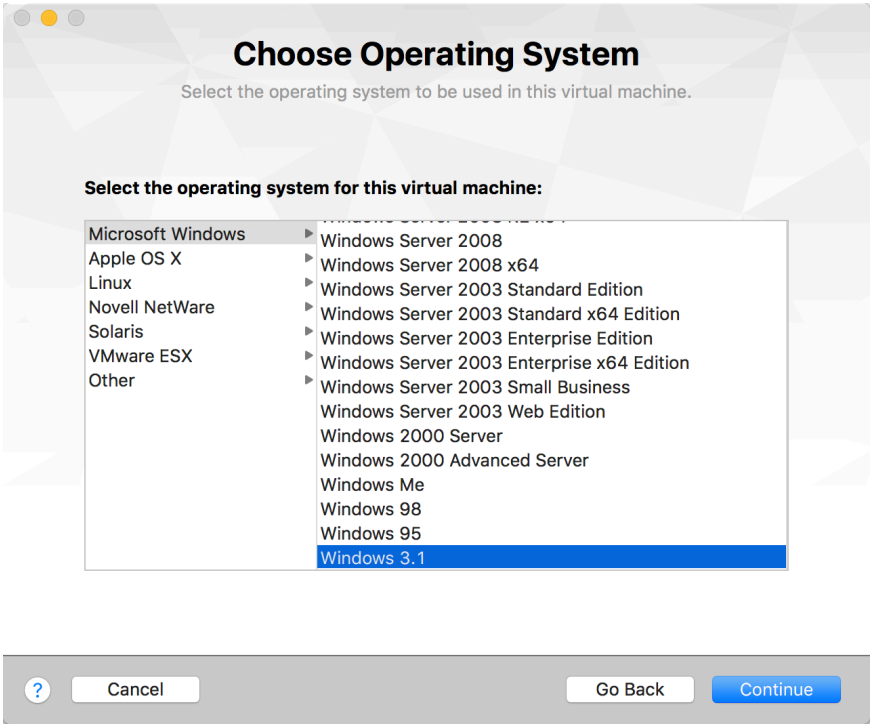
四、实验过程和结果

任务一、创建Dos启动盘

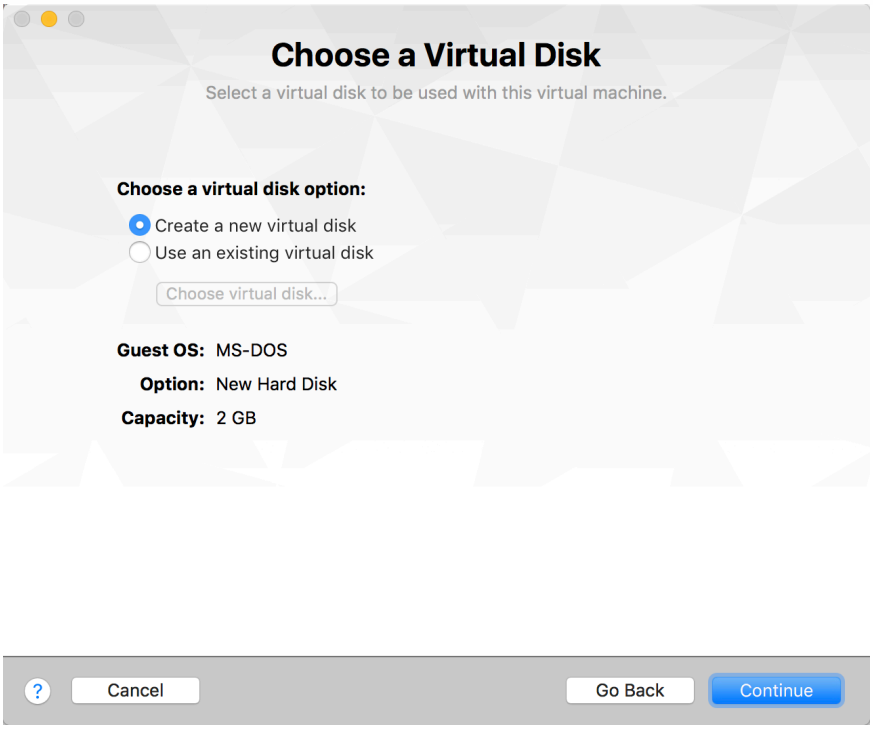
1、首先下载一个dos6.22镜像，使用VMware创建一台dos虚拟机。选择自定义模式



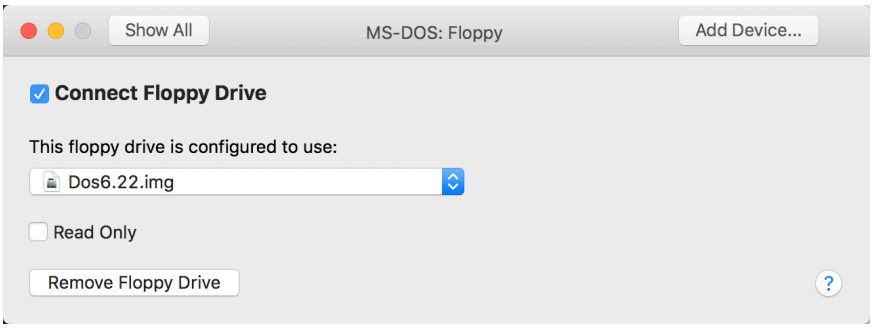
2、选择操作系统，经过尝试，可能是Mac版本的VMware Fusion的bug， 这里如果选择MS-DOS将无法启动虚拟机，选择Windows3.1则可以正常启动。



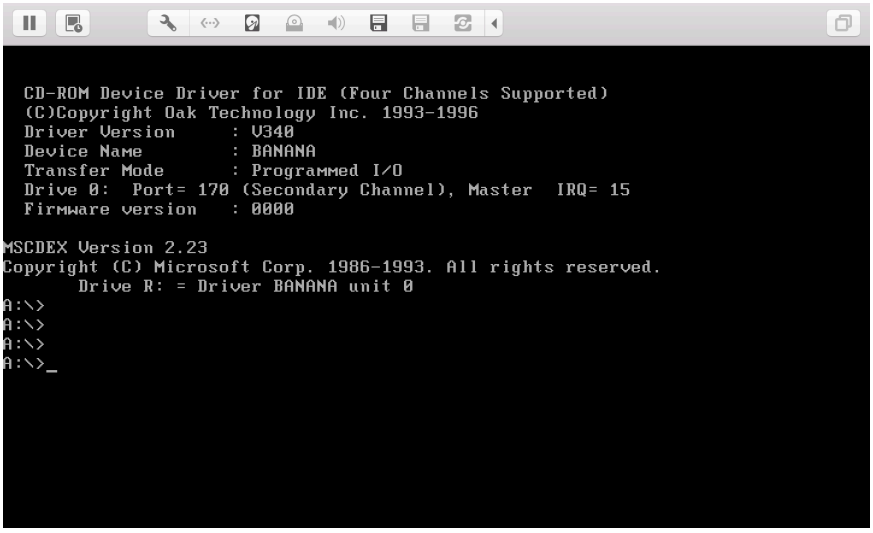
3、硬盘大小按照预设



4、选择自定义配置，添加软盘，选择Dos6.22软盘镜像



5、选择启动介质为软盘，启动虚拟机



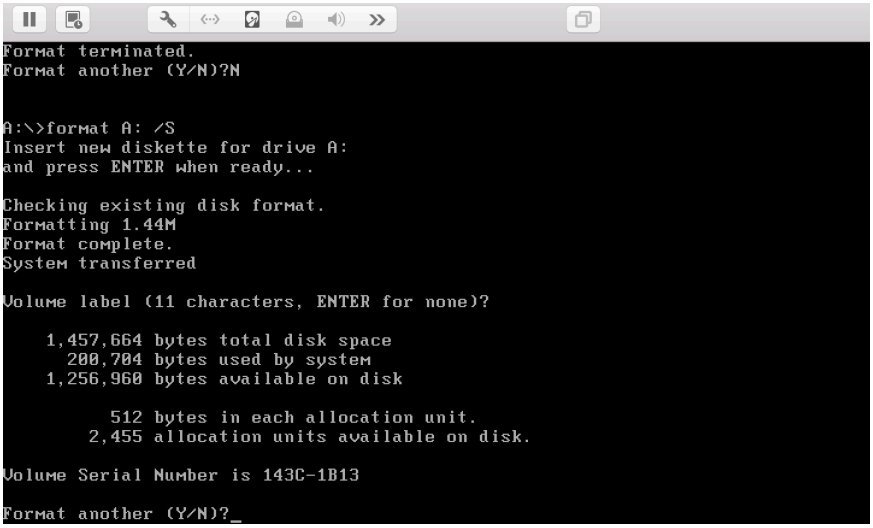
6、在Mac主机上使用bximage工具创建一个空白软盘镜像

```
========================================================================
                                bximage
  Disk Image Creation / Conversion / Resize and Commit Tool for Bochs
         $Id: bximage.cc 13069 2017-02-12 16:51:52Z vruppert $
========================================================================

1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info

0. Quit

Please choose one [0] 1

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] fd

Choose the size of floppy disk image to create.
Please type 160k, 180k, 320k, 360k, 720k, 1.2M, 1.44M, 1.68M, 1.72M, or 2.88M.
 [1.44M]

What should be the name of the image?
[a.img]

Creating floppy image 'a.img' with 2880 sectors

The following line should appear in your bochsrc:
  floppya: image="a.img", status=inserted
```

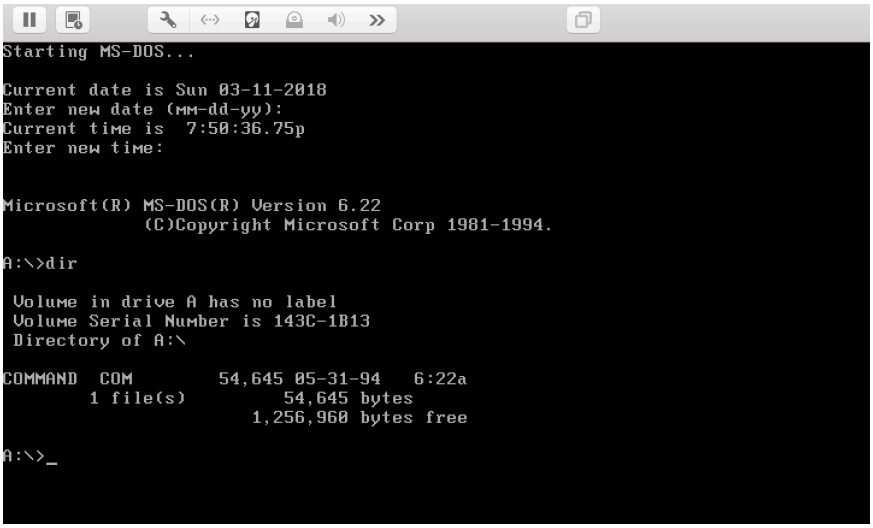
7、在Dos虚拟机中使用format命令创建Dos启动盘

```
format A: /S
```

在执行该命令后，在VMware中移除Dos6.22软盘，插入空白软盘。按下回车继续，就创建好了Dos启动盘。



不拔出启动盘，重启虚拟机，可见该启动盘中只有command.com一个文件，但也可以正常启动系统。



任务二、将软盘首扇区填满你的个人信息

方法一：

这个方法是我首先想到的办法，

首先使用dd工具生成一个512字节的空白扇区

```
dd if=/dev/zero of=floppy.img bs=512 count=1
```

在Mac下，使用Hex Friend编辑器的替换功能，将每18个空白字节替换为"LiXinrui 15323032 "

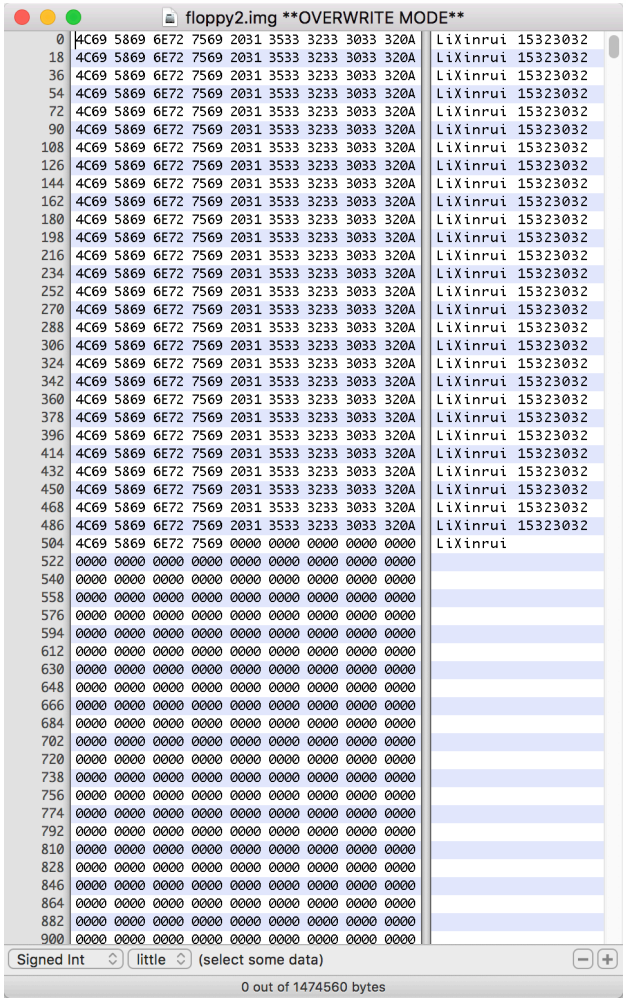
再使用dd工具生成一个1.44Mb的空白软盘

```
dd if=/dev/zero of=floppy2.img bs=1024 count=1440
```

将填满个人信息的扇区替换到该软盘中

```
dd if=floppy.img of=floppy2.img conv=notrunc
```

最终得到首扇区被填满个人信息的软盘



方法二：

这个方法是我学习了nasm语法后想到的做法，使用了nasm的times指令，直接生成了要求的文件。

```
;id.asm
;个人信息字串长为18
times 512 / 18                  db 'Lixinrui 15323032 '
;软盘共1024 * 1440字节。
times 1024 * 1440 - ($ - $$)    db 0
```

下面的shell命令结果显示了它通过编译，且得到了正确的标准软盘大小。内容与刚刚展示的相同，不再重复贴图。

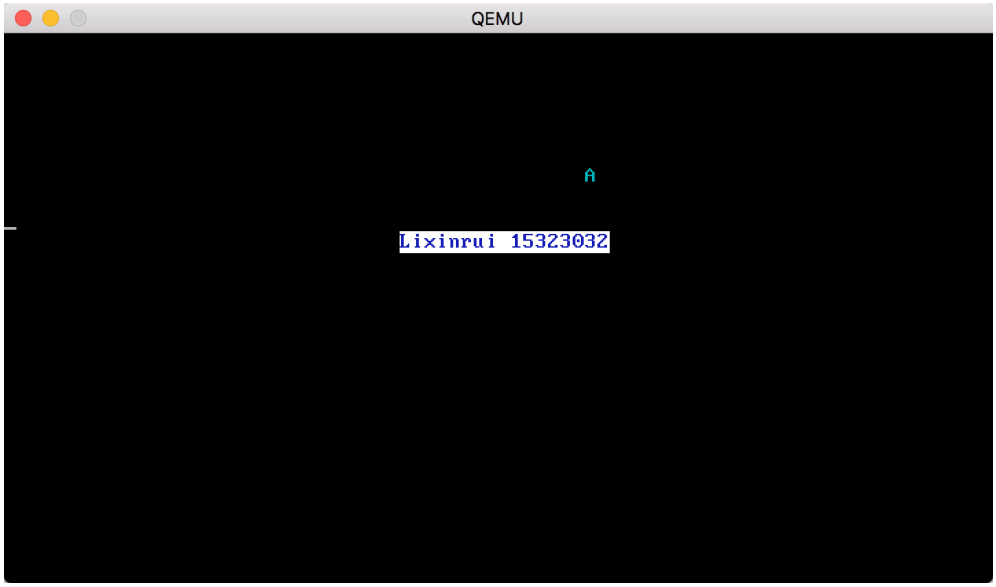
```
lixinrui@lixinruis-MacBook-Pro ~/c/2018-03-11> nasm id.asm
lixinrui@lixinruis-MacBook-Pro ~/c/2018-03-11> ls -l id
-rw-r--r-- 1 lixinrui staff 1474560 Mar 11 22:36 id
```

任务三、设计实验要求的引导扇区程序

在vim编辑器的命令行中直接执行`:make qemu`，整个编译执行过程就进行了起来，vim自动分出窗口显示make命令的输出，可见编译执行无误，切换到桌面便立即看到运行在虚拟机中的程序

```
...作系统/project1 (fish) ⌘1 | ...作系统/project1/test ⌘2 | ...系统/project1/work3 ⌘3 | vim /Users/lixinrui ⌘4 | ...sers/lixir
 0 [1:work3/stoneM.asm]*![2:test/stoneM.asm][5:module.asm]
MiniBufExplorer
 1 ; 李新锐 2018-3
2
 1 org 07c00h                    ;告知编译器代码将被加载到07c00h处
 2
 3 jmp main                      ;从main函数开始执行
 4
 5 ;定义4个函数
 6 ;cls, print_id, change_color, sleep
 7 cls:
 8     pusha
 9     mov ah,0x06
10     mov al,0
11     mov bh,0x07
12     mov ch,0
13     mov cl,0
14     mov dh,24
15     mov dl,79
16     int 0x10
17     popa
18     ret
19
20 print_id:
21     pusha
22     mov ax, myid
23     mov bp, ax       ;es:bp: 字符串首地址
24     mov cx, 17       ;字符串长度
25     mov ax, 01300h   ;调用Write string功能
26     mov bx, 00F1h    ;白底蓝字，闪烁
NORMAL  stoneM.asm
1   || nasm stoneM.asm -o stoneM
 1  || dd if=/dev/zero of=floppy.img bs=1024 count=1440
 2  || 1440+0 records in
 3  || 1440+0 records out
 4  || 1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.003798 s, 388 MB/s
 5  || dd if=stoneM of=floppy.img  conv=notrunc
 6  || 1+0 records in
 7  || 1+0 records out
 8  || 512 bytes copied, 0.0001 s, 5.1 MB/s
 9  || qemu-system-i386 -serial mon:stdio -drive file=floppy.img,index=0,format=raw -smp 1 -m 4
~
~
~
~
~
[:make qemu] [Quickfix List] [-]
```

（编辑环境和一键make）



（截图一，字符A为青色，在个人信息上方）

（截图二，字符A为绿色在图片左边）

五、实验总结

我很开心能够完成本学期操作系统项目的第一个实验。我一直对写操作系统比较感兴趣，大一时有在一位师兄的帮助下做MIT6.828课程的操作系统实验，但因为当时编程能力不足，写到内存分配那里就放弃了。寒假有预习《Orange‘s——一个操作系统的实现》一书，提前搭建好了整个实验环境，学习了org指令和times指令，对本次实验有较大帮助。

本次实验中，有关彩色字符终端（VGA-compatible text mode）我有了一些自己的发现：

我将个人信息的背景色设置为0Fh（亮白色）。然而实验过程中我发现：在qemu虚拟机中个人信息的背景色是亮白色，在bochs虚拟机中却是灰白色，带闪烁，让我很是不解。查询资料才知道：默认情况下，视频硬件设置为将属性位7解释为“前景闪烁”标志，这样前景色只能使用8种颜色，而许多现代BIOS则通过禁用字符闪烁启用全部16种背景色。这个说法也和我了解到的qemu虚拟机是更为现代的虚拟机软件相符合。

相对于C语言，我使用汇编语言编程的经历相对较少。因此在编写代码的过程中，我有先用C语言作为描述程序算法的伪代码。这就让我把新遇到的MIPS编程问题转化为了C语言编程问题和C语言向汇编代码转化问题，而前者是我已经熟练掌握的，后者已经总结好了可以方便地查到，问题就得到了简化。这种转化问题的思路也是我将来可以在其他任务中用到的。

六、参考文献

[1]. MIT 6.828: Operating System Engerning, https://pdos.csail.mit.edu/6.828/2017/

[2]. 《Orange'S:一个操作系统的实现》,于渊, 电子工业出版社, 2009-6

[3]. Nasm Documentation, http://www.nasm.us/doc

[4]. Screen Attributes, http://webpages.charter.net/danrollins/techhelp/0087.HTM