操作系统实验报告

18 级计科超算 18340208 张洪宾

1 实验题目

加载用户程序的监控程序

2 实验目的

- 了解监控程序执行用户程序的主要工作
- 了解一种用户程序的格式与运行要求
- 加深对监控程序概念的理解
- 掌握加载用户程序方法
- 掌握几个 BIOS 调用和简单的磁盘空间管理

3 实验要求

- 知道引导扇区程序实现用户程序加载的意义
- 掌握 COM/BIN 等一种可执行的用户程序格式与运行要求
- 在自己的电脑上以 1.44MB 软驱引导程序的形式，设计一个能执行 COM 格式用户程序的监控程序
- 在 1.44MB 软驱映像中存储若干个用户程序，设计一种简单命令，实现用命令交互执行其中几个用户程序
- 编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

4 实验方案

4.1 硬件或虚拟机配置方法

- 操作系统：macOS Catalina 10.15.4
- 虚拟机：VirtualBox
- 软盘工具:Hex Fiend
- 终端：zsh
- 编译工具：NASM 2.14.02(On macOS)

4.2 设计出四个有输出的用户可执行程序

这次实验对用户程序的要求与上次实验的要求差不多，但是新增了一点就是要求用户分别在屏幕的 1/4 区域运行。

因为我上次重构代码的时候并没有完全采纳老师的方式，而是自己添加了边界变量，所以我这次修改用户程序的时候只修改了边界数值，这也极大地减轻了我的工作量。

由于要求在屏幕的四个 1/4 区域显示我的个人信息，所以我也适当地修改了一下跳动的个人信息：

输入为 1 时，左上的 1/4 屏幕跳动的信息是我的学校和学院的信息“SYSU-SDCS”，

输入为 2 时，右上的 1/4 屏幕跳动的信息是我的学号：“18340208”，

输入为 3 时，左下的 1/4 屏幕跳动的信息是我的姓名拼音缩写：“zhanghb”，

输入为 4 时，右下的 1/4 屏幕跳动的信息是我的邮箱：“2285075600@qq.com”。

于是最后的结果是，根据输入来运行我的用户程序，然后我们就应该看到相应的信息在相应的 1/4 屏幕上跳动。

4.3 设计出软盘的结构

软盘的结构决定了将程序从软盘读取到内存的方式，所以应该提前设计好软盘的结构。

程序	在软盘的起始扇区	占据的扇区数量	功能	载入内存的位置
引导程序 监控程序	0	1	引导程序打印提示信息， 监控接收用户的有效按键， 加载用户程序并跳转执行	07C00h
用户程序 1	1	1	在屏幕左上角使个人信息跳动	0A100h
用户程序 2	2	1	在屏幕右上角使个人信息跳动	0A100h
用户程序 3	3	1	在屏幕左下角使个人信息跳动	0A100h
用户程序 4	4	1	在屏幕右下角使个人信息跳动	0A100h

4.4 设计引导程序和监控程序

由于这次实验难度不是很大，我将引导程序和监控程序合并在一个文件中。并且为了方便输出字符，我使用了宏来在指定坐标输出指定长度的字符串（使用了 10h 号 BIOS 调用）。以下是引导程序和监控程序的详细内容：

4.4.1 引导程序

首先是引导程序。其实引导程序非常简单，与实验一并没有特别大的差别，只是将实验一的个人信息跳动转变成为打印提示信息，然后就应该跳转到监控程序。

如果引导程序和监控程序的大小之和大于 512B，这时候我们就需要把监控程序放到其他扇区，由引导程序将其加载到内存中去。不过由于我们这次的实验比较简单，所以我就没有这么做，而是采取了合并引导程序和监控程序来减小工作量。除此之外，也可以考虑在引导程序处将用户程序载入内存，这样子就可以极大地简化监控程序的实现，只需要不断地在内存中寻址，而不用反复地从软盘中加载程序到内存中。不过此处我实现的时候没有想那么多，直接在监控程序里面根据输入不断地加载用户程序到内存中，反复读取软盘中的用户程序。这一点在实验总结里面会重点讲到。

这样一来，由于引导程序和监控程序都在第一个扇区，在启动虚拟机到时候已经自动载入内存，所以当引导程序打印完相应的字符之后，就可以进入监控程序了。

以下是引导程序的关键代码：即打印提示信息：

```
1 org  7c00h
2
3 %macro print 4  ; string, length, x, y
4     mov ax, cs
5     mov ds, ax
6     mov bp, %1
7     mov ax, ds
8     mov es, ax
9     mov cx, %2
```

```
10      mov ah, 13h
11      mov al, 00h
12      mov bh, 00h
13      mov bl, 07h
14      mov dh, %3
15      mov dl, %4
16      int 10h
17  %endmacro
18
19  section .text
20  begin:
21      call cls;清除屏幕信息
22      print msg2, msglen2,5,35
23      print msg3, msglen3,6,29
24      print msg, msglen,15,8
```

4.4.2 监控程序

根据我的设计，监控程序应该可以读入键盘的有效输入：字符 1、2、3、4 来选择对应的用户程序，然后将对应的程序从软盘中载入内存。

将软盘中的程序载入内存的方法是通过 13h 号的 BIOS 调用。而检测键盘输入的方法则是使用键盘 I/O 中断调用 (16h 号的 BIOS 调用)

当键盘输入的时候，程序会先判断我们的输入的 ASCII 码是否在‘1’到‘4’之间，不是则跳回等待输入的阶段，是的话则将相应的程序加载到内存的 0A100h 的位置，然后跳转到 0A100h 处，执行相应的用户程序。

具体实现的关键代码如下：

```
1   input:
2       mov ah, 0
3       int 16h
4       cmp al, '1'
5       jl input
6       cmp al, '4'
7       jg input
8       mov [chosse], al
9       call cls
10      print msg1, msglen1, 0, 14
11      print chosse, 1, 0, 22
12
13      mov cl, [chosse]
14      sub cl, '0'-1  ;根据输入确定扇区的位置
15      mov ax, cs
16      mov es, ax
```

```
    mov ah, 2
    mov al, 1
    mov dl, 0
    mov dh, 0
    mov ch, 0
    mov bx, OffSetOfUserPrg
    int 13H
    jmp OffSetOfUserPrg
```

4.5 修改用户程序，使之相应键盘输入的 Ctrl+C 来回到监控程序

用户程序是一个无限循环的过程，我们需要输入 Ctrl + C 退出用户程序。而这里可以同样使用上面检测键盘输入的方法检测 Ctrl + C。

而 Ctrl + C 对应的键盘键入的值为 2E03h，所以我们可以在用户程序中加上这样一段程序，在打印完后，对键盘扫描但不等待，如果检测到输入为 Ctrl + C，则跳转回 07C00h，重新执行最开始的程序，即打印提示信息并准备进入新的用户程序。

检测 Ctrl + C 并做出相应反应的代码如下：

```
check:
    mov ah, 01h
    int 16h
    jz Loop
    mov ah, 00h
    int 16h
    cmp ax, 2e03h          ; 检测 Ctrl + C
    jne Loop
    jmp click

click:
    mov cl,1
    mov ax, cs
    mov es, ax
    mov ah, 2
    mov al, 1
    mov dl, 0
    mov dh, 0
    mov ch, 0
    mov bx, instruction
    int 13H
    jmp instruction
```

4.6 生成 zhbOS.img 软盘

我修改了一下上次的 Makefile，如下：

```
IMG = zhbOS.img
PRO = loader.com a.com b.com c.com d.com

all:    pro img clean


img:$(PRO)
ifeq ($(IMG), $(wildcard $(IMG)))
    rm $(IMG)
endif
    mkfile -n 2880b $(IMG)
    dd if=loader.com of=$(IMG) conv=notrunc
    dd if=a.com of=$(IMG) seek=1 conv=notrunc
    dd if=b.com of=$(IMG) seek=2 conv=notrunc
    dd if=c.com of=$(IMG) seek=3 conv=notrunc
    dd if=d.com of=$(IMG) seek=4 conv=notrunc

pro:

%.com : %.asm
    nasm $< -o $@

.PHONY:clean
clean:
    rm -rf *com
```

图 1: Makefile

可以看到，先用 nasm 命令对源文件进行编译，再用 mkfile 命令生成 1.44M 的空白软盘，然后用 dd 命令将生成的文件写到对应的扇区，最后删除中间文件。

5 实验过程

5.1 编译和烧盘过程

进入文件所在的目录，输入 make，生成 zhbOS.img。

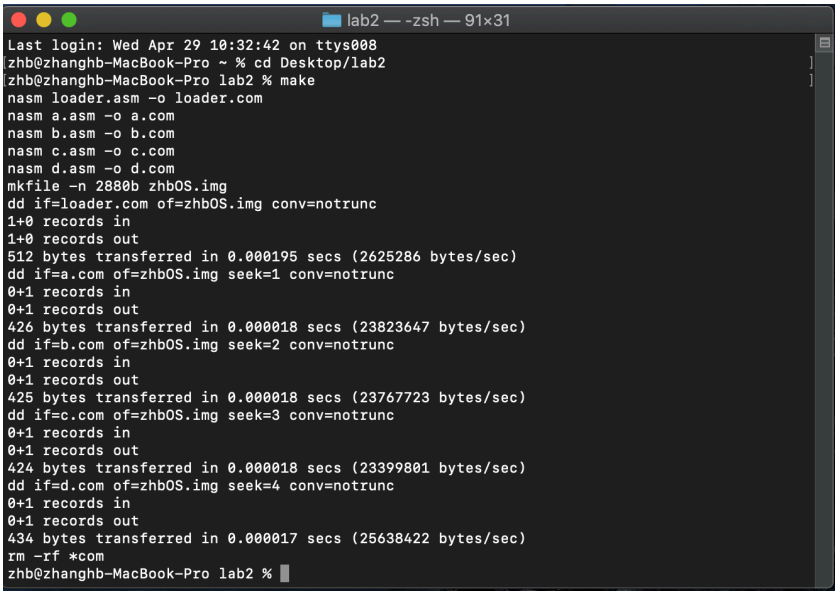

图 2: 输入 make 进行编译和烧盘

得到 zhbOS.img。

5.2 将软盘装载到虚拟机并运行

打开上次创建的虚拟机，在设置中修改存储，将生成的 zhbOS.img 装载到虚拟机上：

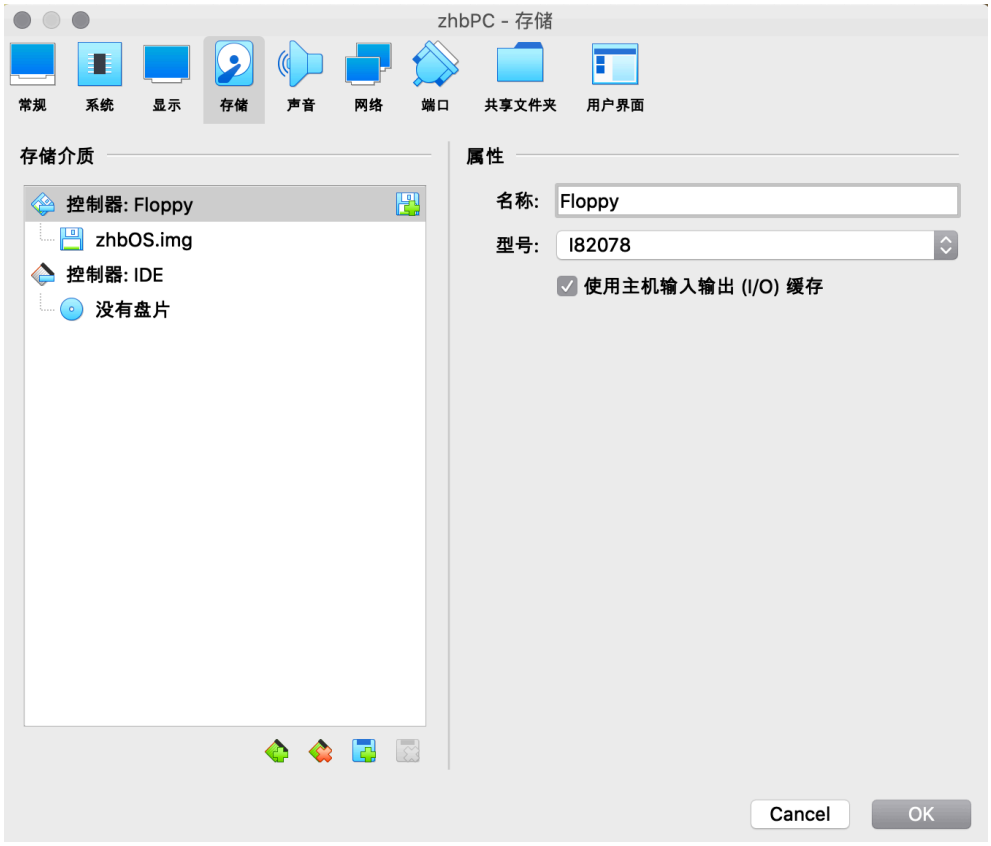

图 3: 将生成的 zhbOS.img 装载到虚拟机

开启虚拟机，可以看到引导程序打印出的提示信息：

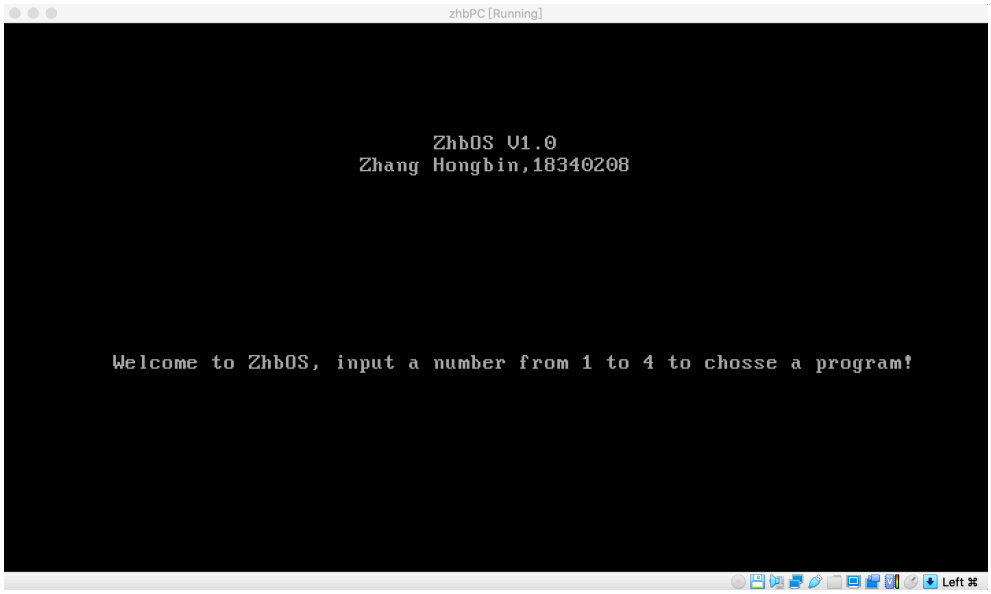

图 4: 引导程序打印完提示信息，并进入了监控程序

根据提示信息，输入 1，2，3，4 可以运行相应的用户程序：

输入 1 时，我的学校和学院信息在屏幕的左上角跳动（与实验一类似），在这里随机截取两个时间点的图来说明：

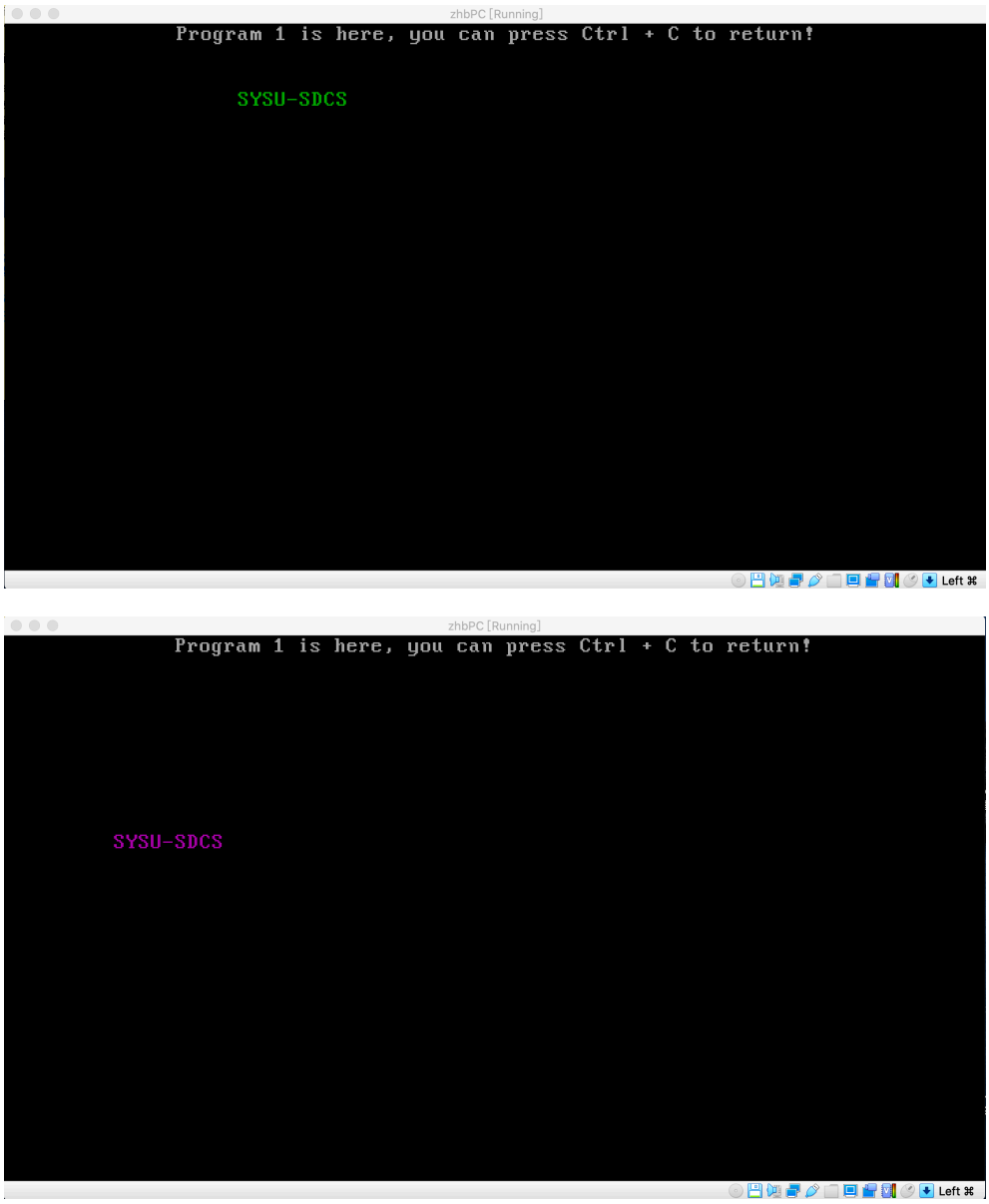

图 5: 输入为 1 时随机截取两个时刻的画面

可以看到输入为 1 时，字符串“SYSU-SDCS”在屏幕上跳动。根据屏幕上方的提示键入 Ctrl + C 可以终止用户程序 1，回到监控程序处（与图 3 情况相同）。

再输入 2，我的学号在屏幕的右上角跳动，在这里随机截取两个时间点的图来说明：

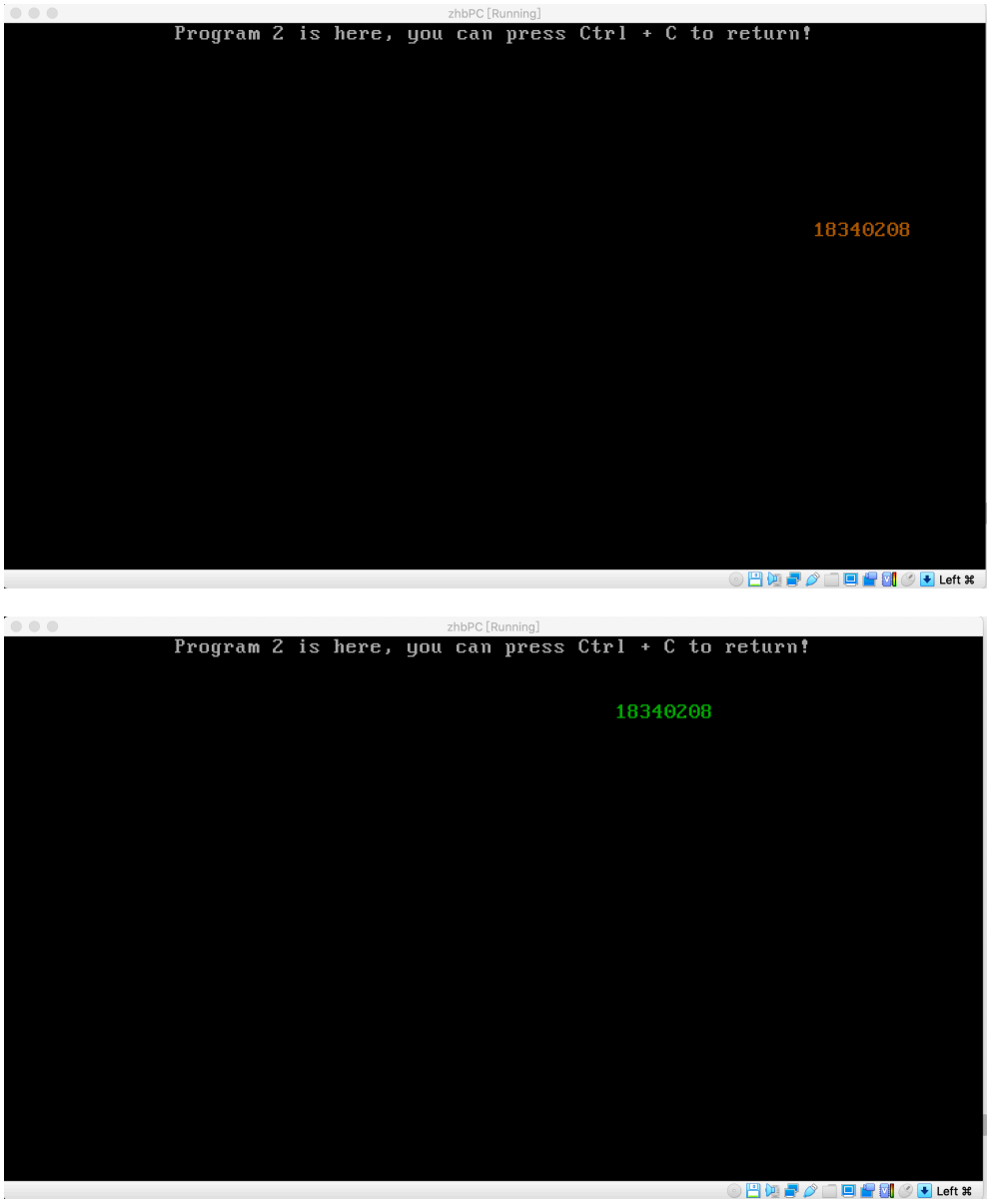

图 6: 输入为 2 时随机截取两个时刻的画面

键入 Ctrl + C 回到监控程序（与图 3 情况相同），然后输入 3，可以看到我的姓名缩写在屏幕的左下角跳动：

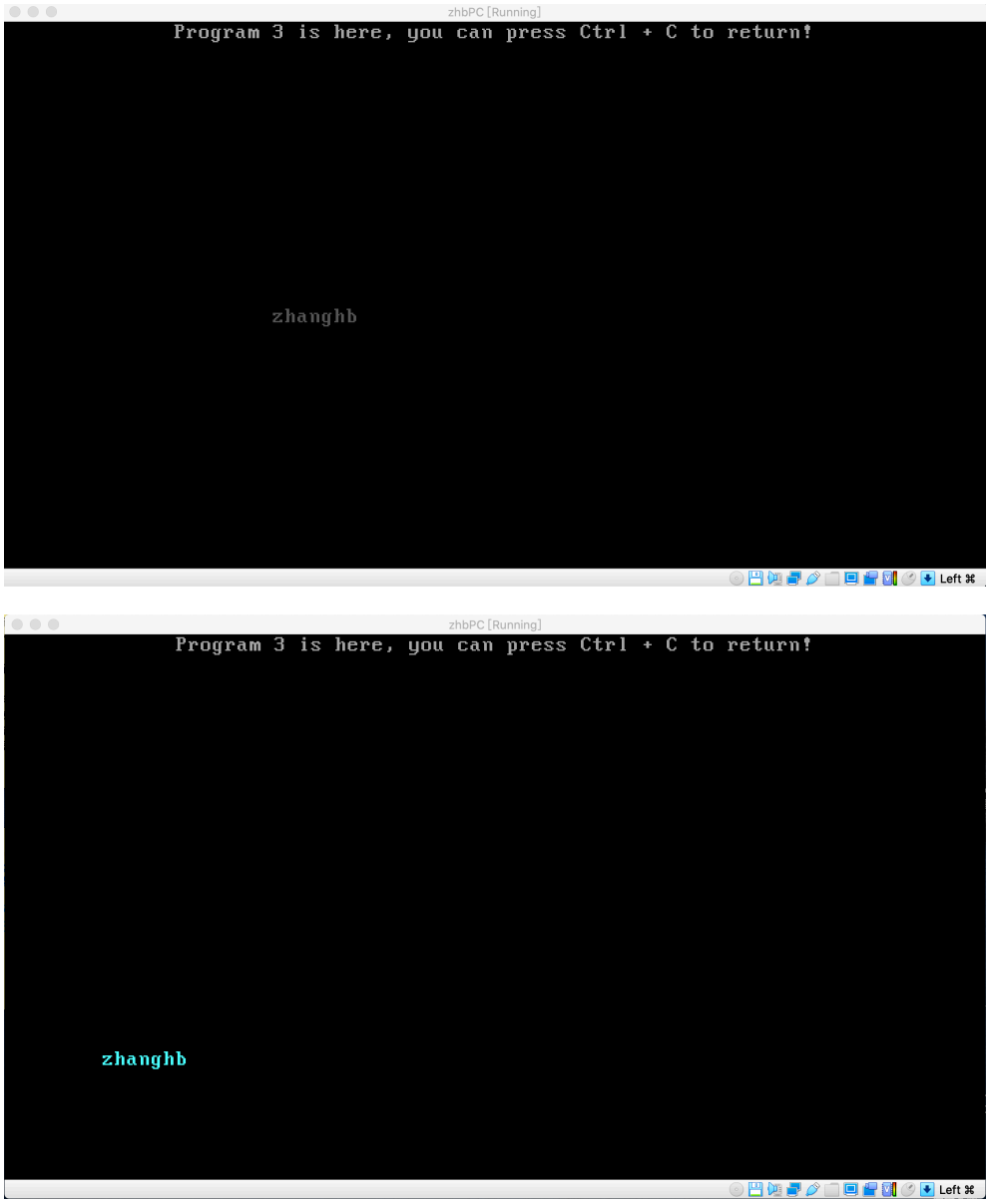

图 7: 输入为 3 时随机截取两个时刻的画面

键入 Ctrl + C 回到监控程序（与图 3 情况相同），最后输入 4，可以看到我的邮箱在屏幕的右下角跳动：

图 8: 输入为 4 时随机截取两个时刻的画面

由于截图不能完整地展示我的程序的运行情况，所以我在文件夹中附带了一段录屏，更加完整地展示了程序的运行情况。

6 实验总结

6.1 遇到的问题与解决问题时的细节

- 首先最开始我没有理解好 org 的含义，没有给我的用户程序加上对应的 org，所以出了比较大的问题，当我从监控程序跳转到用户程序的时候不能够输出字符串。

- 对 BIOS 调用也不够熟悉，最开始我在用户程序中，使用键盘 I/O 中断调用时功能号为 ah=0, 然后发现用户程序会卡在第一次输出处，无法继续运行，除非按下键盘的任意键才可以打印下一个输出。通过上网查询我了解到 16h 号 BIOS 调用的阻塞和非阻塞的情况：在引导程序处检查输入的时候，应该采用阻塞型的，令 ah=0，调用 16h 号 BIOS 调用，等待键盘的输入。而在用户程序检测键盘输入是否为 Ctrl+C 的时候，则需要使用非阻塞型的，令 ah=1，调用 16h 号 BIOS 调用，才可以使得程序一直执行下去。

- 在输出的时候不想要再像实验一一样逐个循环把字符串放到显存中了，于是使用了 10h 号 BIOS 调用。并使用了宏，将输出字符串的过程进行封装，便于之后的操作。

- 此次实验我将引导程序和监控程序合并在一个文件，事实上这是不太科学的，因为实际上当监控程序稍微大一点的时候，引导程序和监控程序的大小之和就会超过 512B，这样子的话计算机启动的时候就不能正确读取监控程序。所以在下一次实验我将把引导程序单独放在第一个扇区。

- 这次我将四个程序都加载到内存中的同一个位置，当每次切换程序的时候用 16h 号 BIOS 调用将目标程序加载到 0A100h 处，这样子其实也比较不符合现代计算机设计的理念。我觉得可以用引导程序把用户程序加载到内存中到 4 个不同到位置，就可以直接在内存中跳转，而不用去考虑从软盘中读取程序，这样子可以减小从软盘中读取程序花费的时间，比较符合现代计算机设计的局部性原理。下次实验我也会直接在引导程序把用户程序载入内存中的不同位置。

6.2 心得体会

这次实验是直接建立在上一次实验的基础上的，并学习了一些 BIOS 调用。在这个过程中也发现了操作系统实验的环环相扣，也更加注重了代码的可读性和结构性。若代码过于凌乱则易出现到后期需要重构的现象，这对于以后实验的发展非常不利。

另外虽然这次实验比较简单，但是还是有很多细节值得思考，通过这个实验，我对理论课上学到的知识有了进一步的理解，也更加理解了操作系统运行的方式。