一、实验目的

- ☒ 实现原始的批处理操作系统
- ☒ 学习内核从外存加载和结束用户程序的方法
- ☒ 学习使用BIOS中断

二、实验要求

- ☒ 实现监控程序，显示必要的提示信息后，从引导盘的特定扇区加载一个他人开发的COM格式的可执行
- ☒ 设计四个有输出的用户可执行程序，分别在屏幕1/4区域动态输出字符，如将用字符‘A’从屏幕左边
- ☒ 修改参考原型代码，允许键盘输入，用于指定运行这四个有输出的用户可执行程序之一，要确保系统
- ☒ 自行组织映像盘的空间存放四个用户可执行程序。

三、实验环境

与实验一大致相同：

主机操作系统：Mac OS 10.12

编辑器：Vim 8.0.1400、VS Code 1.21.0

汇编器：Nasm 2.13.02

虚拟机、调试器：Bochs 2.6.9

版本控制：Git 2.15.1

自动构建：GNU Make 3.8.1

四、实验方案

（一）、基础要求部分

1、监控程序的启动

与实验一相同：监控程序放入软盘第0号扇区，并将扇区最后两个字节写入0x55，0xAA后，即可被BIOS加载
0x10中断打印出开机欢迎字符，并提示用户输入。监控程序使用BIOS
0x16号中断的0号功能（AH=0）阻塞读取键盘缓冲区，获得用户输入，根据输入判断要加载哪个用户程序。

2、加载用户程序

(1) 编写20h，21h号中断，设置PSP

监控程序在加载用户程序必须做相关操作，而不能像提供的示例代码那样直接jmp到用户程序中，才能确保

一个较为简便的方法是使用call指令，call指令能够将当前指令的下一条指令的CS、IP压栈，在用户程序中调

因此，我选择使用了真实的DOS系统的解决方法：在加载用户程序前，首先在用户程序载入地址（本实验中
20h（int 20h的机器码），在0x0A处写入返回地址的IP、CS。

在真实的DOS系统中，程序有两种方法返回DOS：一是通过ret指令返回到PSP中的int
20h指令，二是调用给你21h中断的4ch功能。这两个中断都是DOS系统中断，在我的操作系统中我对它们进

```
interrupt_20h:                      ; 20h    21h
      mov ah, 4ch
interrupt_21h:
      cmp ah, 4ch                   ;    4ch
      jnz panic_21h_func_not_impl   ;    kernel panic
      jmp dword[0xA00A]             ;   jmp psp
      iret
panic_21h_func_not_impl:            ;   4ch
      print_string panic_21h_msg, paini_21h_len, 0, 0
      jmp $
```

通过这种方法，我的操作系统拥有了从正确的DOS程序返回的兼容性。

(2)安装中断

中断向量表是从内存0号单元开始1k字节的内存空间，最多存放256个中断服务程序的入口地址，安装N号中
0000:[4N + 2]处。

(2) 加载用户程序到内存并跳转

实现加载用户程序到内存使用的是BIOS 13H号中断的2号功能：分别将驱动器、柱面、磁头、扇区号写入dl,
ch, dh, cl寄存器，在ah中写入读取的扇区数量，并在bx中写入要把扇区载入到的内存地址，调用中断即可实

在本实验中，驱动器、柱面、磁头号均为0，　　第N个用户程序被放在第2N
+　　1个扇区，并最多占用两个扇区，因此设置al为2，载入的地址是bx　　=
0xA100。

最后，监控程序首先使用pusha保护了当前寄存器值，然后通过jmp指令启动了用户程序。

3、用户程序（屏幕1/4区域弹射字符）的实现

(1)响应键盘输入

用户程序需要在正常执行过程中响应键盘输入，随时准备返回操作系统。这一功能是通过BIOS的0x16中断的

(2)屏幕区域划分

本实验使用了NASM的macro功能，将4个弹射字符程序共同的部分写为一个macro，这个macro接收4个参数

(3)返回操作系统

本实验中按下ESC键（27）会使得用户程序退出。上文已经提到，本操作系统中的用户程序可以使用标准的
20h指令或调用int 21h的4ch功能）。

（二）、扩展创新部分

1、代码改进和《贪吃蛇》游戏的实现

本实验中，我首先改进了实验一中的弹射字符程序的算法，改为使用当前速度方向Vn和当前位置Pn来确定

$$P_{n+1} = P_n + V_n (1)$$

使用该算法，实验一中的程序缩减了80行的代码。

除此之外，还实现了显示固定长度的字符长串功能。这一功能是通过使用数组保存之前字符的位置，结合r

最后，在能够显示固定长度的字符长串的基础上，我制作了《贪吃蛇》游戏。游戏中玩家能够控制一个初

这里用到的随机数程序是原理是，通过调用BIOS 0x1A号中断，读取当前时钟计数。然后与
0x11ee 相与运算后（根据本实验实际需要计算出的值，防止除法商太大溢出）使用div指令与数字N进行运算
1的随机数。

2、自动处理任意多用户程序的Makefile

在实验二中有4个以上的用户程序要编译，写入磁盘镜像，逐一手动写入的方法显得耗时耗力，更为不可取

改进部分的代码如下：

```
include gmsl                                    # plus
disk_index = 1                                  #
SHELL=/bin/bash
AS = nasm
ASFLAG = -f bin
user_src = $(sort $(wildcard user*.asm))        #
user_bin = $(user_src:.asm=.o)                  #
user%.o : user%.asm common.asm                  #
    $(AS) $(ASFLAG) $< -o $@
kernal_src = myos1.asm
kernal_bin = $(kernal_src:.asm=.o)
kernal : $(kernal_src)
    $(AS) $(ASFLAG) $(kernal_src) -o $(kernal_bin)
floppyfile = disk.img
```

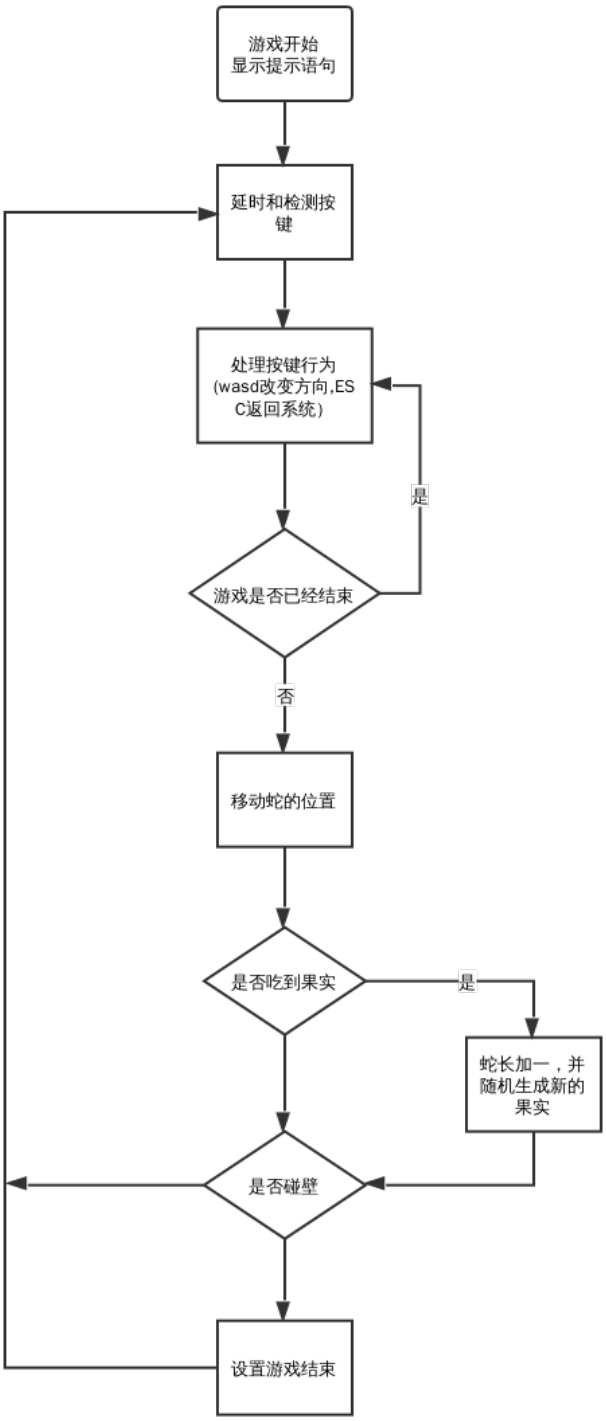

Figure 1: Untitled Diagram

```
clean_disk:                                     #
    dd if=/dev/zero of=$(floppyfile) bs=512 count=2880
write_kernal: clean_disk kernal                 #
    dd if=$(kernal_bin) of=$(floppyfile) conv=notrunc
define DO_write                                 #
dd if=$(strip $(1)) of=$(floppyfile) bs=1024 seek=$(strip $(disk_index)) conv=notrunc
$(eval disk_index = $(call plus,$(disk_index),1))

endef
write_all_progs: write_kernal $(user_bin)    # foreach
    $(foreach user_prog, $(user_bin), $(call DO_write, $(user_prog)))
```

四、实验过程和结果

在VS Code编辑器中写好监控程序myos1.asm和user1.asm到user5.asm五个用户程序后，直接在屏幕下方的内置
make bochs，各个程序的编辑，写入软盘的过程就很快完成了，bochs虚拟机立即加载软盘镜像启动了虚拟机

（图一：编辑和运行环境）

在VS Code内置Terminal中输入c使虚拟机继续执行，操作系统首先进入了监控程序界面：

（图二：监控程序界面）

输入数字1到5可以选择执行用户程序，执行完一个程序后按下ESC回到监控程序并切换到下一个。

1到4依次是在屏幕左上到左下顺时针4个区域弹射固定长度的字符长串的程序，下面4张图片将依次演示：

（图三：在左上角运动的用户程序一）

（图四：在右上角运动的用户程序二）

（图五：在左下角运动的用户程序三）

（图六：在右下角运动的用户程序四）

最后是展示《贪吃蛇》游戏截图，另有游戏录像在screen_record文件夹中。

（图七：当前蛇长度为3）

（图八：当前蛇长度为7）

五、实验总结

本周的操作系统实验花费了我不少的心血，学到了不少的新东西。在开始研究和编写代码之前，我首先翻看
Make文档。然而因为Makefile的语法过于复杂，我又结合网上搜寻，花费了很大功夫才写出全自动化构建所

本次实验代码量相对于第一次实验要大很多，我通过宏和模块化的方法，使得整个编码过程还算顺利，然而

本次实验中我还借rand函数的实现练习了C函数调用在汇编中的形式。虽然在之前的程序设计和计算机组成

本次实验成功后，我还把操作系统写入U盘，在物理机上引导执行了。看到一台真实的电脑运行着自己的操

六、参考文献

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

[2]. GNU Make Documentation, https://www.gnu.org/software/make/manual/

[3]. Phoenix BIOS 4.0 User’s Manual, http://www.esapcsolutions.com/ecom/drawings/PhoenixBIOS4_rev6UserMan.p

[4]. Ascii Table, https://www.asciitable.com/

[5]. PSP - DOS Program Segment Prefix Layout, http://stanislavs.org/helppc/program_segment_prefix.html

[6]. C Calling Convention and the 8086, http://ece425web.groups.et.byu.net/stable/labs/StackFrame.html