不可以使用BIOS的中断

需要自己实现

在保护模式下完成

需要段描述符等信息

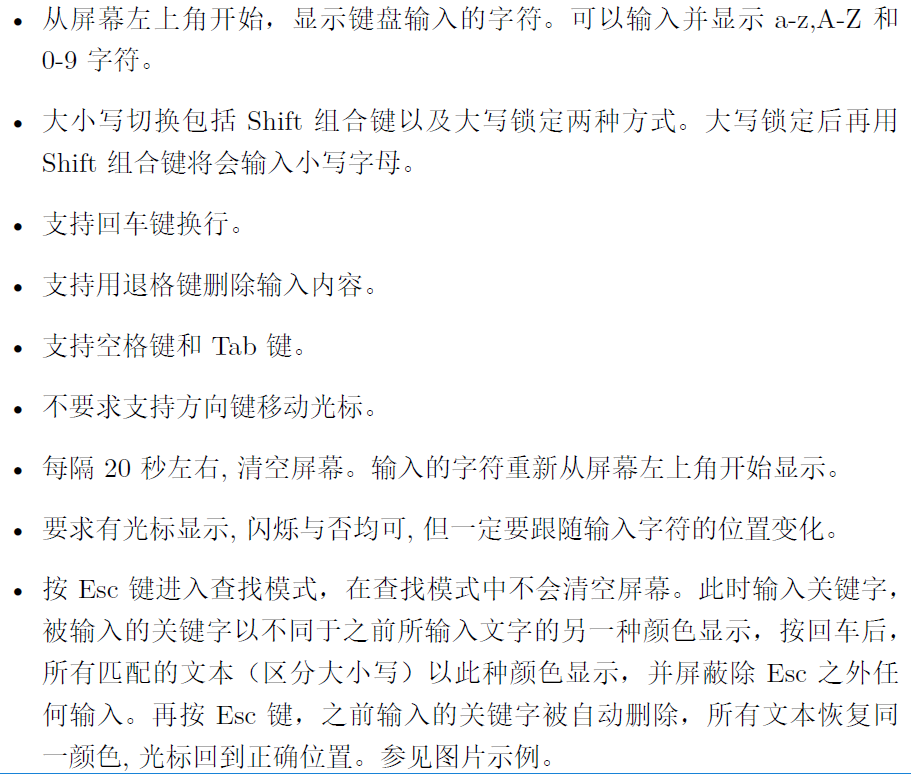
支持make run

书第3.4 节, 5.5 节, 7.1到7.3 节。

boot.bin,loader.bin 和kernel.bin在其基础上修改

boot.bin 和loader.bin, 你可以直接使用《Orange’s》的代码,

## 需求





       本次实验中tab键需实现以下功能：

        1. 对齐，相当于位置模4，输出（4-余数）个空格

        2. 被视为整体，即使用tab键输出了3个空格的空间，则删除时应当回退3个空格

        3. 关于“换行”特殊情况：是指在本行使用tab键+回车后，再使用退格键（删除回车）、退格键（删除tab键的输出）

查找思路：

1）先取得查询串，从查询模式开始到当前位置的显存中取出字符串，存到char数组里。

2）将0至查询模式开始的字符串从显存中取出，存到全局变量中。

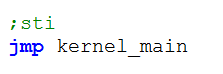
3）对目标字符串进行查找，每找到一个匹配的字符串，就将它开始和结束位置传给一个用于重设颜色的函数，将该子串重新设置颜色，如此循环直至查询完毕。

4）当查询模式结束时，将所有字符串存储的全局变量写回到显存中。

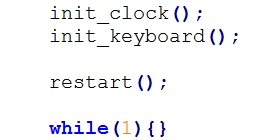
## 理解框架代码

boot.asm和loader.asm可以不去考虑，只知道它们最后转向了kernel.asm。

在kernel.asm中调用了C函数

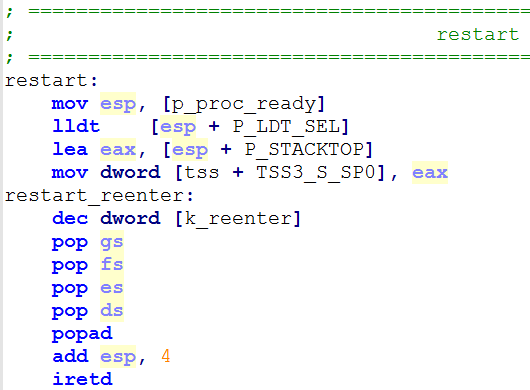


这个函数定义在main.c中，至此后面的内容都是C编写的。



kernel\_main函数中初始化了键盘和时钟的中断，然后调用restart

restart定义在kernel.asm中



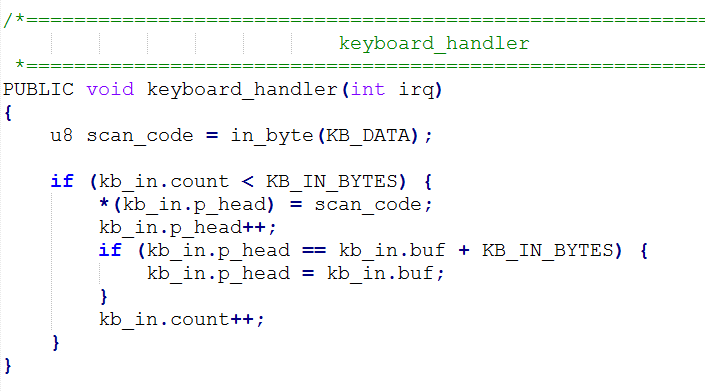
看不懂在干嘛，可能也是初始化吧。

之后开始空转。

如果有键盘和时钟中断，那么会转向相应的中断处理程序。

我们要做的就是实现键盘和时钟的中断处理程序。

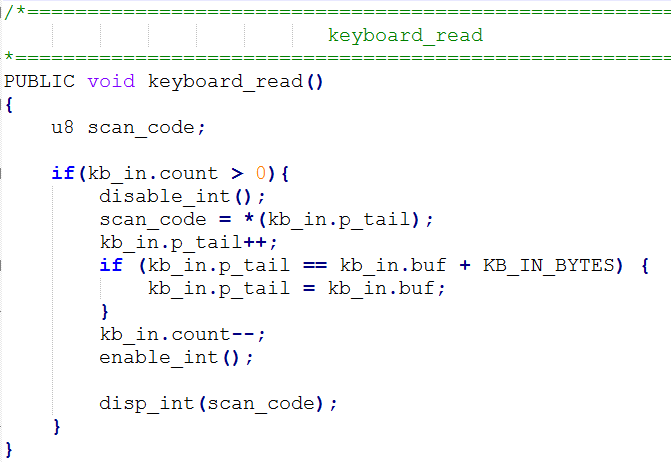
键盘的中断处理程序放在keyboard.c中



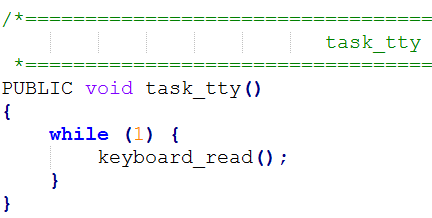
keyboard\_handler函数是在每次接收键盘中断时被调用的，作用是从8042的端口读取一个字节，然后放入缓冲区中，缓冲区大小为32个字节。

**现在有一个函数disp\_int可以供我们调用，用于显示一个数字/字符。**

**还有一个disp\_str，也可以调用，是传入一个char\*，把字符串打印出来。**

**（下面仅是把扫描码打印出来，不做任何处理）**

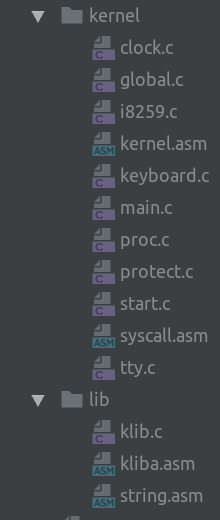
keyboard\_read函数是如果缓冲区存在字符，那么从缓冲区读出一个字符，然后从缓冲区中将其去掉，最后显示该字符。



而keyboard\_read函数是在task\_tty函数中被调用的，

**disp\_pos是一个全局变量，记录了当前显示器显示字符的位置**

disable\_int和enable\_int是用于关中断和开中断的函数，是汇编的C封装。



string.asm中提供了三个函数的实现：

void\* memcpy(void\* es:p\_dst, void\* ds:p\_src, int size);

void memset(void\* p\_dst, char ch, int size);

char\* strcpy(char\* p\_dst, char\* p\_src);

klib.asm中提供了以下函数的实现：

void disp\_str(char \* info);

void disp\_color\_str(char \* info, int color);

void out\_byte(u16 port, u8 value);

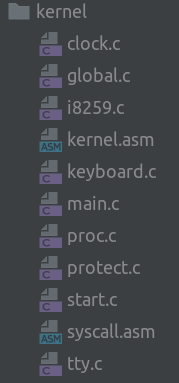
u8 in\_byte(u16 port);

void disable\_irq(int irq);

void enable\_irq(int irq);

void disable\_int();

void enable\_int();



clock.c提供了时钟中断的处理器实现

PUBLIC void clock\_handler(int irq)

这个函数在每次时钟中断时都会被调用

keyboard.c 提供了键盘中断的处理器实现

PUBLIC void keyboard\_handler(int irq)

这个函数在每次键盘中断时都会被调用，作用是将8048中缓冲区的字符取出来，放入到自己的缓冲区中。

PUBLIC void keyboard\_read()

这个函数是tty.c中的task\_tty函数循环调用的

PUBLIC void task\_tty()

{

while (1) {

keyboard\_read();

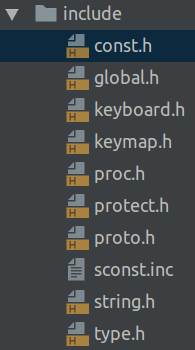
}

}

keyboard\_read函数是进行一些逻辑判断，最后把code交由tty.c中的PUBLIC void in\_process(u32 key)函数来进行打印

键盘处理方面，我们主要编写的是keyboard\_read函数和in\_process函数

下面看一下头文件们



关注一下

/\* klib.asm \*/

PUBLIC void out\_byte(u16 port, u8 value);

PUBLIC u8 in\_byte(u16 port);

PUBLIC void disp\_str(char \* info);

PUBLIC void disp\_color\_str(char \* info, int color);

/\* protect.c \*/

PUBLIC void init\_prot();

PUBLIC u32 seg2phys(u16 seg);

可以看出它把项目中的大部分的public函数都在此做了声明

我们可能会用到的是keyboard.h和keymap.h以及const.h

keymap.h中定义一个全局的数组，可以通过make code取出对应的字符

u32 keymap[NR\_SCAN\_CODES \* MAP\_COLS] = {...}

keyboard.h中定义一些特殊的按键，是人为将特殊按键赋予一些自定义的码

const.h中有一些与显示器有关的常量

/\* VGA \*/

#define CRTC\_ADDR\_REG 0x3D4 /\* CRT Controller Registers - Addr Register \*/

#define CRTC\_DATA\_REG 0x3D5 /\* CRT Controller Registers - Data Register \*/

**#define START\_ADDR\_H 0xC /\* reg index of video mem start addr (MSB) \*/**

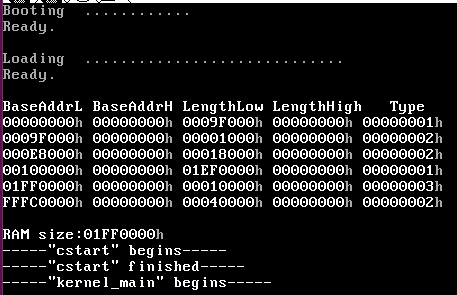
**#define START\_ADDR\_L 0xD /\* reg index of video mem start addr (LSB) \*/**

**#define CURSOR\_H 0xE /\* reg index of cursor position (MSB) \*/**

**#define CURSOR\_L 0xF /\* reg index of cursor position (LSB) \*/**

**#define V\_MEM\_BASE 0xB8000 /\* base of color video memory \*/**

#define V\_MEM\_SIZE 0x8000 /\* 32K: B8000H -> BFFFFH \*/



boot.asm调用load.asm，loader.asm调用kernel.asm，kernel.asm调用main.c的kernel\_main

init\_clock();

init\_keyboard();

restart();

它调用了clock.c的init\_clock函数，调用了keyboard.c的init\_keyboard函数

调用了kernel.asm的restart函数

## 知识点

### 键盘

键盘控制器8042

键盘编码器8048

敲击键盘产生的是扫描码，分为make code（按下）和break code（弹起）。

每一个按键都对应一个make code和一个break code。

当8048检测到一个按键时，会将扫描码发送给8042,8042将它转为scan code set1 扫描码，将其放入输入缓冲区，然后8042告诉8259A产生中断IRQ1。如果此时有新的按键，那么8042不会接收，直至输入缓冲区被清空。

从8042的输入缓冲区读取：从 0x60端口，读一个字节 in al,0x60

扫描码表

<http://www.computer-engineering.org/ps2keyboard/scancodes1.html>

break code = make code | 0x80

make code = break code & 0x80

如何判断是make code 还是break code：和0x80做与运算，如果结果为0，那么是make code，不为0，是break code。

### 显示器

一个屏幕可以显示80行，25列，每个字符占2个字节，高字节存放颜色，低字节存放ASCII码。

一个屏幕占用4KB的显存。

设置光标相关：

VGA系统有6组寄存器，与光标有关的是CRT Controller Registers

有两个端口，一个对应着数据寄存器，一个对应着地址寄存器。

先往地址寄存器写入寄存器的索引号，再向数据寄存器写入一个新值，就可以将该数据寄存器的值更新。

### 时钟

  时钟中断控制器采用的intel的芯片 8253，它有三个16位的计数器，其中**counter0对应端口地址是0x40h，控制寄存器对应0x43h**。其中counter0输出到IRQ0，每隔一段时间产生依次时钟中断。计数器的原理是这样的，有一个输入频率1193180HZ,每次输入将导致计数器-1，当计数到0时，产生中断。而16b的循环需要65535次，所以时钟中断的默认发生频率是1193180/65536=18.2HZ.

55ms ticks+1

20\*1000/55 = 3636

if(ticks % 3636 == 0){ 已经过去了20s}

## makefile注意点

在64位系统中编译32位程序需要修改Orange’s的makefile中的以下部分

makefile

ASM = nasm

DASM = ndisasm

CC = gcc

LD = ld

ASMBFLAGS = -I boot/include/

ASMKFLAGS = -I include/ -f **elf32**

CFLAGS = -I include/ -c **-m32** -fno-builtin **-fno-stack-protector**

LDFLAGS = **-m elf\_i386** -s -Ttext $(ENTRYPOINT)

DASMFLAGS = -u -o $(ENTRYPOINT) -e $(ENTRYOFFSET)