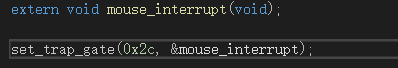
鼠标驱动和简单的图形接口实现

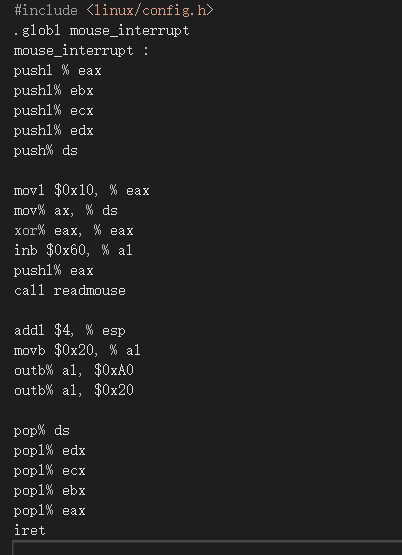
一、捕获鼠标中断

1. 在kernel/chr\_drv/console.c添加中断，0.11中的unistd.h中也要加宏



2. kernel/chr\_drv/keyboard.S

​ 从鼠标设备的数据寄存器中将鼠标信息读取出来（通过0x60端口)，通过inb命令将该数据寄存器中存放的1B信息读入到AL中，之后通过readmouse函数进行具体处理。



1. i8042有两个I/O端口：命令端口0x64，数据端口0x60。

1. ​ 鼠标涉及到的命令：

1. 0xA8

允许鼠标操作

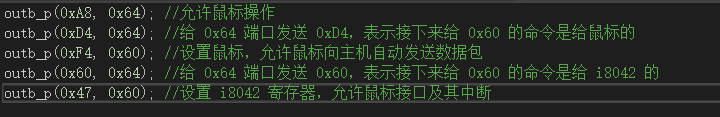
2. 0xD4

告诉i8042,发往0x60端口的参数是给鼠标的

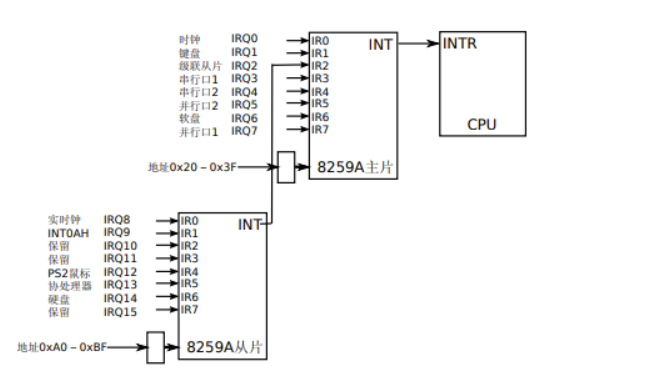
3. 0x60

将发往0x60的参数写入i8042(0100 0111,即0x47)

2. 设置i8042(在set\_trap\_gate(0x2c,&mouse\_interrupt)之前)

 3. 绑定0x2c与鼠标请求中断IRQ12

boot/setup.s 中8259A屏蔽了所有中断（ICW1所有位置1），应解开

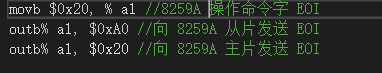




0xEF 对应的二进制是 1110 1111，是将从片 IR4 的屏蔽打开。以后，鼠标中断请求会经过从片的 IR4 和主片的 IR2 到达 CPU。

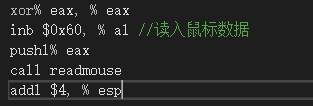
4. 结束中断以便接受下一次中断

kernel/chr\_drv/keyboard.S

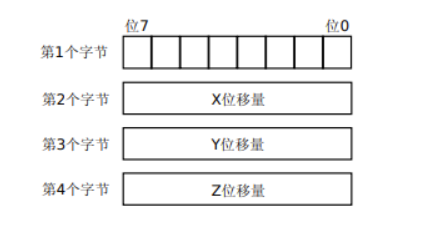


二、 鼠标输入数据的解码

1. 将将鼠标输入数据 从 0x60 端口读进来，将其压栈后再调用 C 函数 readmouse



2. 鼠标输入数据的格式



| 二进制位 | 含义与作用 |

| -------- | ------------------------------------------------------- |

| 7 | Y 溢出标志位，1 表示 Y 位移量溢出 |

| 6 | X 溢出标志位，1 表示 X 位移量溢出 |

| 5 | Y 符号标志位，1 表示 Y 位移量为负（向上为正、向下为负） |

| 4 | X 符号标志位，1 表示 X 位移量为负（向左为负、向右为正） |

| 3 | 保留位，总为 1 |

| 2 | 中键按下标志位，1 表示按下了中键 |

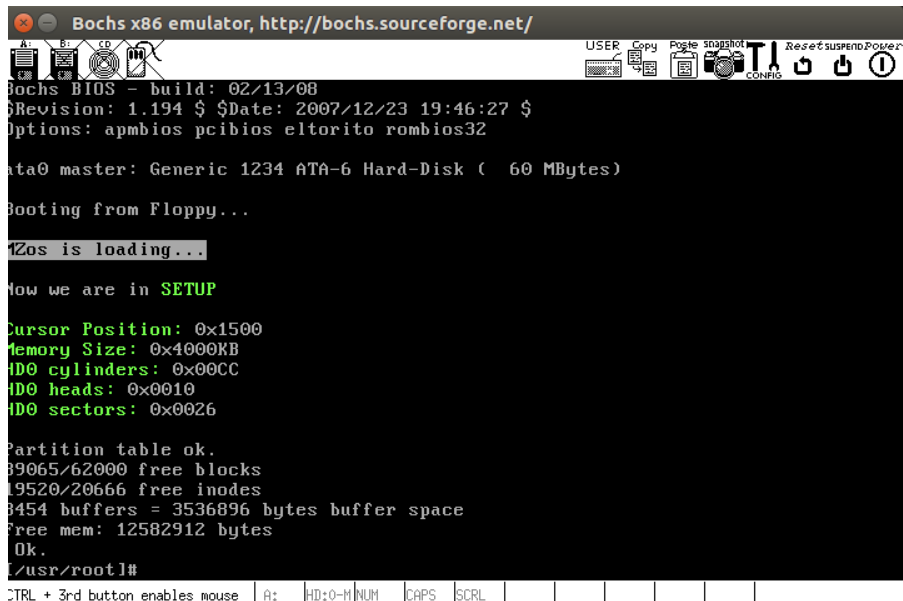
| 1 | 右键按下标志位，1 表示按下了右键 |

| 0 | 左键按下标志位，1 表示按下了左键 |

3. 二位鼠标数据解码函数readmouse

kernel/chr\_drc/tty\_io.c (不能放在console.c 中，会因为释放内核页表报错)

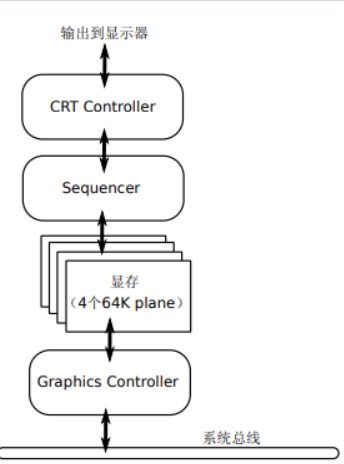
三、阶段成果展示



四、显示器的图形显示模式

1.启用VGA图形模式

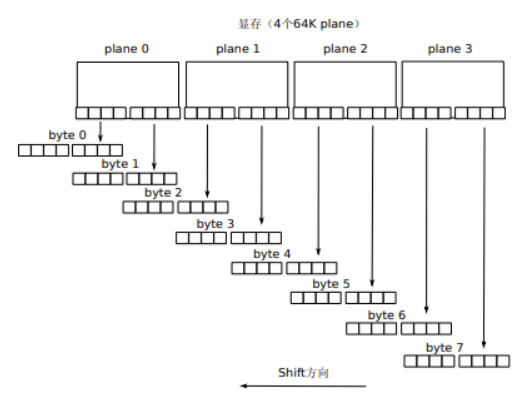
工作模式：



2.建立像素点阵与显存之间的映射

启用mode 0x13,即线性256色模式，分辨率为320\*200，每个像素用一个字节表示，共64000个像素点，64000B，略小于64KB

拼装器工作模式：



将 4 个显存 plane 中 同一个地址处的 4 个字节按照每 4 位一组进行左移，共移出了 8 个字节，显卡会取出Byte1,3,5,7并将这四个 字节依次扫描到屏幕上形成四个像素

3.设置屏幕分辨率

通过CRT控制器完成，工作原理：

0）从左到右，从上到下，通过扫描线扫描到显示器上

1）scanline counter用来计数是第几根扫描线，从0开始则扫描输出第一行像素

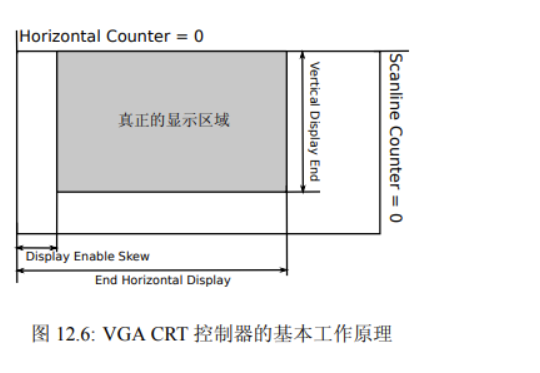
2）在生成每一行像素时，horizontal counter开始工作，其经过Display Enable Skew设定的距离后开始输出像素，当其增加到End Horizontal Display后CRT停止从显存中读取数据。

3）当一行输出完成后，horizontal counter置0，而scanline counter是否增加取决于SLDIV,默认每隔两个扫描周期后加一

4）扫描线增加1后，显存的shift地址向后移动一行对应的字节数Offset Register \* MemoryAddresses \*2并重复步骤2

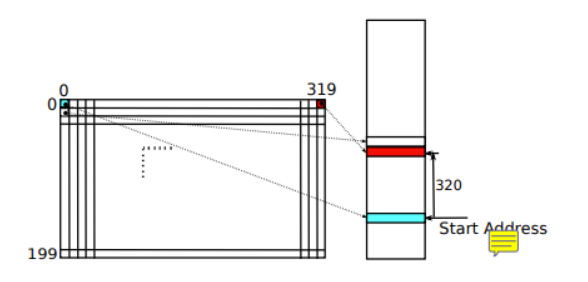
5）当扫描线增加到Vertical Display End后不再输出

原理图：



4.开始绘制屏幕

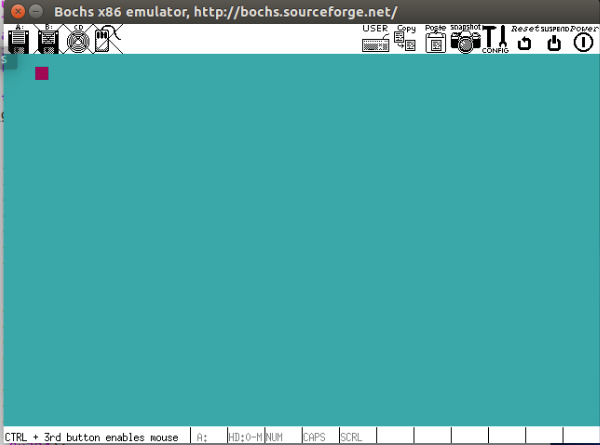
只需将起始地址设为0xA0000即可，现在VGA屏幕中的图像对应内存如下：



5.代码

图形模式初始化通过系统调用sys\_init\_graphicis实现，

6.阶段成果展示：



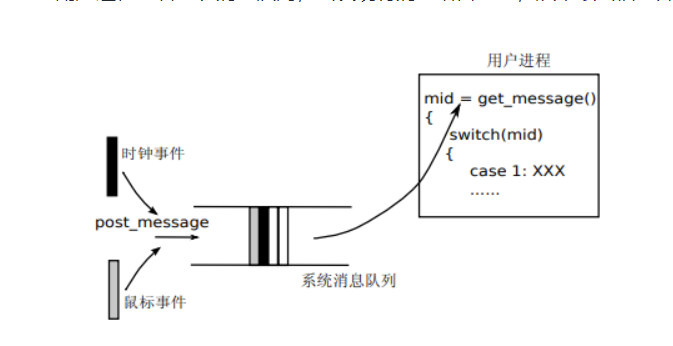
3.消息驱动框架

1.原理

1. OS收集计算机各个地方的事件，如键盘按下，鼠标按下，网卡有数据包，时钟到时等

2. 操作系统负责将各个事件做成一个数据结构并将其放入系统消息队列中

3. 用户进程查看系统消息队列，当发现有消息给自己时，从中取出信息并进行处理

 2.主要工作

1. 定义消息队列数据结构

原计划多进程共享队列，因为目前只有一个进程会使用这个调用，因此将原本计划的msg结构体（含mid,pid,next指针）简化为全局变量jumpp

在include/linux/tty.h处定义全局变量

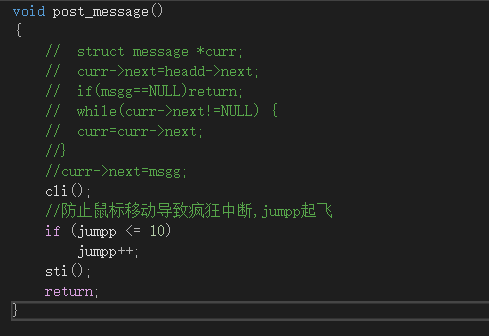
extern int volatile jumpp;

同时在各个调用此变量的文件\*.c处也要定义

int volatile jumpp;

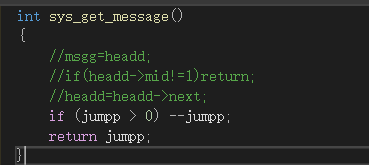
2. 实现post\_message将消息放入系统消息队列中

位于kernel/chr\_drv/ttyio.c



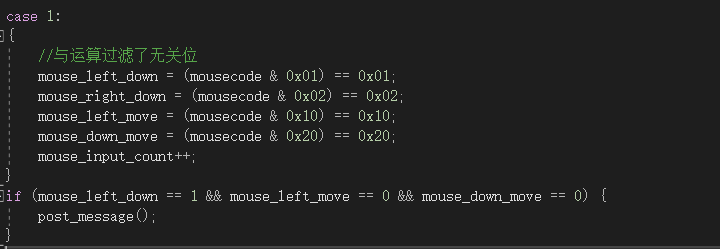
3. 能为用户态进程提供获取系统消息队列的系统调用get\_message,编写对应的内核函数sys\_get\_message

1. 位于kernel/init\_graphics.c

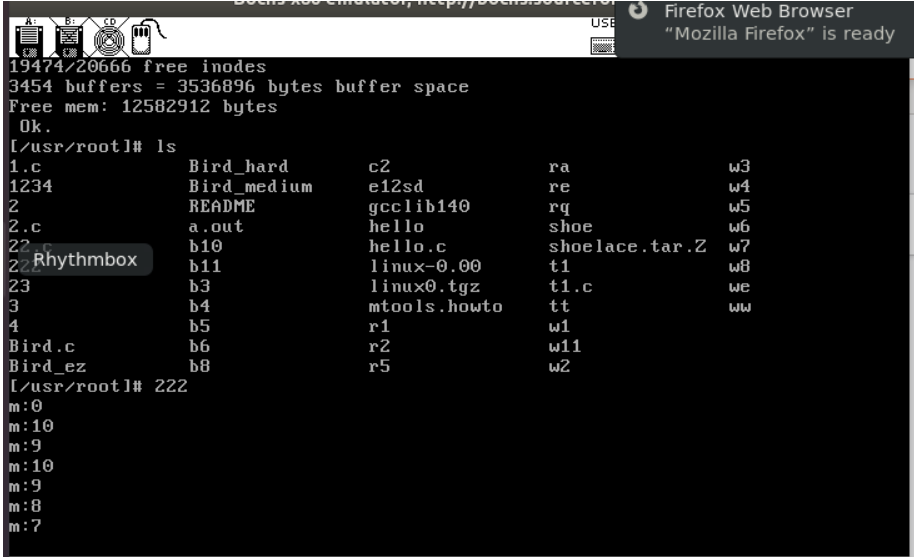


4. 在readmouse函数合适地方调用post\_message

位于kernel/chr\_drc/tty\_io.c



5. 测试





其中m的值就是用户程序调用get\_message的返回值jumpp

4.Flappy Bird

1.思路：小鸟：一直向下移动，鼠标左键点击小鸟会向上移动一定距离；障碍物以一定速度左移，以实现整个场景运动。

1. 创建所有屏幕对象

2. 产生循环，定时重画屏幕

1. 刷新屏幕为蓝，33

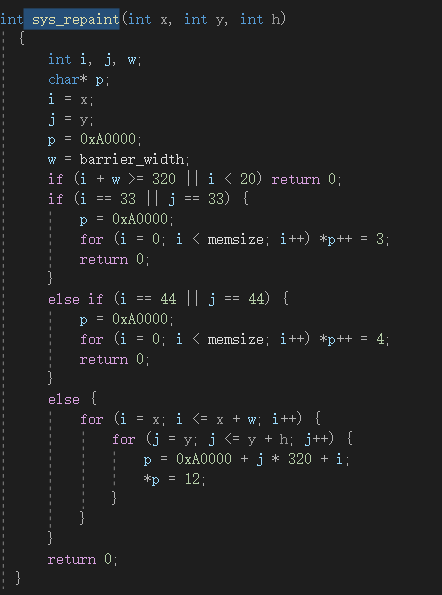
2. 小鸟向下移动,py[0]+=10;

3. 障碍物左移,px[k]-=20;

4. 若有鼠标点击事件则小鸟向上移动py[0]-=10;

5. 判断是否GAME OVER,结束则爆红,44

2.代码（用户程序代码略）



3. 最终效果

