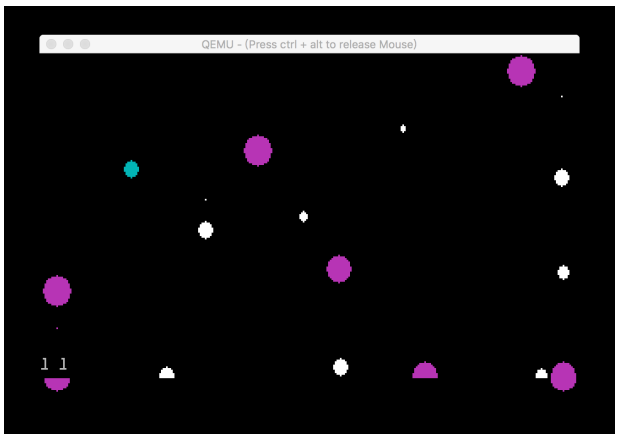
OS Lab 01 引导程序与游戏实验报告

151220030 → 高子腾 ：GZT@outlook.com

2017 年 3 月 24 日

第一部分　实验结果

1. 实验框架搭完
2. 实现 bootloader 并加载游戏
3. 键盘中断和时钟中断的控制
4. 实现 printk 函数用于调试输出



第二部分　实验环境

编译虚拟机环境　Debian (32bit)

GCC 编译版本　gcc version 4.9.2 (Debian 4.9.2-10)

QEMU 运行环境　虚拟机 Debian (32bit) 与 OS X 10.11.6。游戏在虚拟机 Debian 上需降帧运行。在 OS X 则上不需要。

Git 记录　本实验带有 Git 记录。由于实验的初期一直在安装各种宿主虚拟机，所以会出现初期时实验 Author 有多个的记录，都是我。虚拟机内时间比较异常，所以 git log 上的时间也比较异常（好像是落后 6 天）

第三部分　实验过程

文件架构与 Makefile　我所使用的文件架构与 Makefile 是网页上所提供的。此外我还在提供的 Makefile 基础上添加了 make run 和 make image 的目标。make run 是 make qemu 的另一名字。make image 是只编译不运行，添加的目的是在 Debian 里编译完成后直接在 OS X 中 qemu 运行。

mbr 和 bootloader　mbr.S 文件里依次进行关中断，清段寄存器，开启 A20 线，设置 GDT，开启保护模式，在保护模式中设置段寄存器以及设置栈帧的起始位置，然后通过 call bootmain 进入 c 语言的 bootloader.c 的部分。bootloader.c 主要做的事情就是判断 elf 魔数，加载各个 elf 头，然后跳转到 elf 代码载入位置。

游戏前体部分　游戏主体部分在于 kernel.c 和 game.c 中。此处的 kernel.c 并不是真正意义上的内核，也没有内核态和用户态之分。kernel.c 和 game.c 的区分主要是以逻辑作为依据的。kernel.c 为游戏初始化好串口，时钟信号，中断描述符表和初始化中断。其中初始化中断描述符表的代

码来自于 PA 实验修改而得。初始化好这些后，kernel.c 将控制权交由 game.c，game.c 设置好键盘回调函数和时钟回调函数后，开中断，初始化显存，之后进入死循环，等待中断的到来。

游戏主体部分 由追赶时间驱动的代码有一下逻辑，游戏状态机的状态字一开始为 *GAME_START*，绘制欢迎画面后，初始化游戏的主要参数，在标准输出端用 printk 输出"Press Q to start" 后，进入 *GAME_READY* 状态，等待 Q 被按下。一旦 Q 被按下后，进入 *GAME_ING* 状态，游戏的主要逻辑就在此，一旦游戏结束，则进入 *GAME_END* 状态，输出"You're Dead" 后进入 *GAME_START* 状态。

关于 printk printk 的实现由两种状态的状态机来实现。关于十进制和十六进制的输出是由取模，除法一步一步完成的。但是我注意到在 %d 的负号情况下，-2147483648 (0x80000000) 并不能非常好的直接按上面的流程输出，所以对于这个数字，只能直接输出一段字符串了。

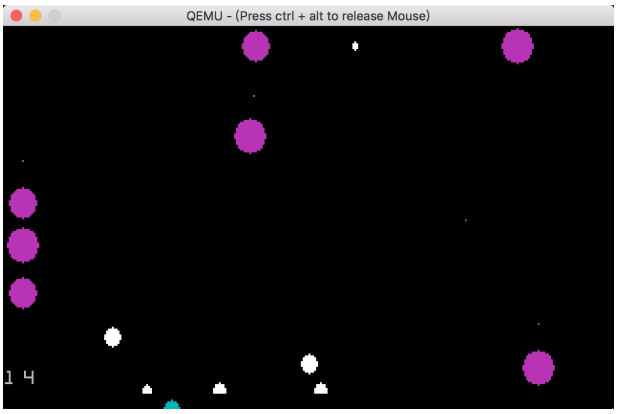
显示逻辑 显示逻辑经过两层抽象，第一层是 video.h, 第二层是 stage.h. video.h 的实现较为丑陋，不过还是往 *VCACHE* 缓存绘制像素点为主，并把绘制的行设置脏位和次脏位，在必要绘制时把 *VCACHE* 的脏行写入显存，并把不是脏行的次脏行清空。*refreshCache* 提供了每个时钟都把脏位清空的方法，而次脏位不会被清空（具体逻辑无法在此具体描述，感兴趣可看代码）这一切是为了提高显示效率，所以有时候在游戏中会出现残留的点，是因为我没有设置每帧都强制清屏，而是间隔清屏。次脏位的清空是由绘制计时决定的。当然这里用到的 *memcpy* 是汇编实现的。stage.h 则是较高层的抽象，能绘制数字，矩形 (和不太那么像的）圆形，还有一张特定的开始画面，在 stage.h 中用到了一个哈希值，可以通过哈希值判断画面是否不同于前一帧，如果相同，则不重绘。

游戏逻辑 游戏的逻辑是用 W, A, S, D 移动小球，吃掉比自己小的白球，不被比自己大的红球吃掉。吃掉白球会长大，球会从画面中心出来。每吃掉一个球游戏加分。还是好玩的。左下角会显示当前的得分。如感觉太快可以把 game.c 中定义的 *QUICK* 注释掉。

第四部分 实验遇到的问题及心得

保护模式输出字符串 我发现一旦进入保护模式，再以实验准备讲义的逻辑输出字符串的时候，就会重启。这个问题应该是字符串的地址在保护模式下不再是汇编里的地址。

QEMU 的显示问题 如图，OS X 的 QEMU 最下端的显示区域经常不被刷新，而 Debian GUI 下就没有这个问题。



时钟驱动显示逻辑? 当直接在时钟回调函数中渲染显示画面时，一旦把时钟信号调为 400HZ 发现，游戏一进入就会重启。回想起 PA 中打字小游戏的逻辑，他是在死循环中渲染画面的，按照这样的逻辑一改，果不其然，可以以 400HZ 运行。对此我的看法是，在时钟回调函数中如果做了超时的事情，下一次时钟信号再次进入，虽然关中断，但还是在 *hlt* 中，会造成错误。而追赶方法可以由 *hlt* 语句来直接管理各种信号，一旦进入渲染逻辑，就可以不管各种信号了。

画圆 由于我的游戏是大球吃小球的类似产物，所以画圆不可避免。而我选择的画圆逻辑是性能最差的开根号法，但是我也是对了开根号的逻辑

做了一定的优化了。然而明显的发现到，当画面中很多大圆时，画面就显而易见的降低了刷新频率。

随机数的生成 由于这是个游戏，不可避免的要用到随机数生成，随机数的生成代码可以在 kernel.c 中找到，利用了线性同余的方法。鉴于游戏的特性，我还用了游戏中玩家的 x, y 坐标以更随机化。