PA4 来自外部的声音
中断与 I/O

姓名： 陈越琦

学号：121160005

院系：计算机系

邮箱：15905193328@163.com

时间：2015/1/20

1.实验环境

ubuntu 14.04.2操作系统
gcc 4.8.2编译器
vim 文本编辑器

2.思考题

异常与函数调用
我们知道进行函数调用的时候也需要保存调用者的状态: 返回地址, 而进行异常处理之前却要保存更多的信息. 尝试对比它们, 并思考两者保存信息不同是什么原因造成的.

异常处理程序和源程序是相互独立的。在异常处理程序运行过程中,EIP,CS,EFLAGS 与通用寄存器会被用到。故为了异常处理结束后能返回到源程序继续执行,需要保存 EIP、 CS、EFLAGS 与通用寄存器的值等进程上下文。而函数调用还是一个进程,不用担心 CS,EFLAGS 与通用寄存器的值发生变化对后面的程序继续运行,故只需保存返回地址。

特殊的原因
我们在上一小节中提到: "由于一些特殊的原因, 这三个寄存器的内容必须由硬件来保存". 究竟是什么特殊的原因, 使得 EFLAGS, CS, EIP 三个寄存器的值必须由硬件来保存? 尝试结合 IA-32 中断机制思考这个问题.

一方面，如果使用软件保存容易保存错误的值,导致异常处理结束后无法恢复到我们想要的场景继续执行。另外一方面，使用硬件来保存可以提高指令运行速度。

重新设置 GDT
为什么要重新设置 GDT? 尝试把 init_cond()函数中的 init_segment()注释掉, 编译后重新运行, 你会发现运行出错, 你知道为什么吗?

因为此时段描述符发生了变化,全局描述符表也发生了变化。在 start.S 中设置的原全局描述符表不能再用于后面的程序,因而需要重新加载 GDT。

不可缓存(uncachable)的内存区域

我们知道使用 cache 可以提高访问内存的速度, 但是对于用作内存映射 I/O 的内存区域, 使用 cache 却可能造成致命的错误. 因此一般会将这部分内存设置为不可缓存, 这样, 每一次对它们的访问都不会经过 cache, 而是老老实实地访问相应的"内存区域". 你知道为什么要这样做吗?

如果是 I/O 外设，这些内存范围如果使用 cache。当设备发生变化时,相应的 cache 不会发生变化,故访问 cache 而非相应内存区域时会发生致命的错误。

理解 volatile 关键字

也许你从来都没听说过 C 语言中有 volatile 这个关键字, 但它从 C 语言诞生开始就一直存在. volatile 关键字的作用十分特别, 它的作用是避免编译器对相应代码进行优化. 你应该动手体会一下 volatile 的作用, 在 GNU/Linux 下编写以下代码:

```
void fun() {
	volatile unsigned char *p = (void *)0x8049000;
	*p = 0;
	while(*p != 0xff);
	*p = 0x33;
	*p = 0x34;
	*p = 0x86;
}
```

然后使用-O2 编译代码. 尝试去掉代码中的 volatile 关键字, 重新使用-O2 编译, 并对比去掉 volatile 前后反汇编结果的不同.

你或许会感到疑惑, 代码优化不是一件好事情吗? 为什么会有 volatile 这种奇葩的存在? 思考一下, 如果代码中的地址 0x8049000 最终被映射到一个设备寄存器, 去掉 volatile 可能会带来什么问题?

因为被 volatile 修饰的变量可被不同线程访问和修改。如果没有 volatile 关键字,则编译器可能优化读取和存储,可能暂时使用寄存器中的值,如果这个变量由别的程序更新了的话,将出现不一致的现象。在此例中去掉 volatile 可能会出现设备寄存器的内容在期望之外被修改的问题。

如何检测多个键同时被按下

你应该从数字逻辑电路实验中认识到和扫描码相关的内容了: 当按下一个键的时候, 键盘控制器将会发送该键的通码(make code); 当释放一个键的时候, 键盘控制器将会发送该键的断码(break code), 其中断码的值为通码的值+0x80. 需要注意的是, 断码仅在键被释放的时候才发送, 因此你应该用"收到断码"来作为键被释放的检测条件, 而不是用"没收到通码"作为检测条件.

在游戏中, 很多时候需要判断玩家是否同时按下了多个键, 例如 RPG 游戏中的八方向行走, 格斗游戏中的组合招式等等. 根据键盘扫描码的特性, 你知道这些功能是如何实现的吗?

根据键盘扫描码的特性,可以根据收到通码和断码的先后顺序来判断是否有多个按键同时被按下,是哪几个按键被同时按下。

提高磁盘读写的效率

阅读 kernel/driver/ide/disk.c 中的代码, 理解 kernel 读写磁盘的方式. 以读操作为例, 你会发现磁盘中的每一个数据都先通过 in 指令读入到寄存器中, 然后再通过 mov 指令把读到的数据放回内存; 写操作也是类似的情况. 思考一下, 有什么办法能够提高磁盘读写的效率?

使用 DMA 控制 I/O。主存与设备的数据交换不需要经过 cpu。

3.总结

因为要准备期末考试，考完试之后又一心想着回家，实在无心写 PA4。胡乱做了一点，就放下了。所以整个 PA4 基本相当于没有动。

不过回顾整个 PA 的过程，收获还是很大的。学会如何调代码，学会如何使用 makefile 工具，对 C 语言有更加深入的理解。关键是，写 PA 对于理解 ICS 课程内容帮助很大，把一些虚的知识落到实处，还是很有成就感的。

最后，祝老师和助教新春快乐！