7 实验六 中断与系统调用

7.1 实验目的

1. 加深对设备管理基本原理的认识，了解键盘中断、扫描码等概念。
2. 掌握 Linux 使用系统调用的基本原理。

本实验基于 Linux 0.11 源码。

7.2 中断处理

中断处理主要涉及两个代码文件：asm.s 和 traps.c 文件。

中断信号通常可以分为两类：**硬件中断**和**软件中断**（异常）。每个中断由 0-255 之间的一个数字来表示。软件中断是由 CPU 执行指令时探测到异常情况而引起的，通常可分为**故障**（fault）和**陷阱**（trap）两类。

在进程将控制权交给中断处理程序之前，CPU 会首先将至少 12 字节的信息压入中断处理程序的堆栈中。

由于有些异常引起中断时，CPU 内部会产生一个出错代码压入堆栈（异常中断 int 8 和 int 10-14），而其他中断并不带有这个出错代码（例如除零出错和边界检查出错）。因此对中断的处理，需要根据是否携带出错码分别处理，但处理流程是相同的。

1. 所有寄存器入栈。
2. 出错代码入栈。无出错代码时，使用 0。
3. 中断返回地址入栈。
4. 所有段寄存器置为内核代码段的选择符值。
5. 调用相关 C 处理函数。
6. 弹出入栈的出错码和后来入栈的中断返回地址。
7. 弹出所有入栈寄存器。

8. 中断返回。

其中，调用的 C 函数在 kernel/traps.c 中实现。压入堆栈的出错代码和中断返回地址是用作 C 函数的参数。

7.3 系统调用

Linux 应用程序调用内核的功能是通过中断调用 int 0x80 进行的，寄存器 eax 中放调用号。因此该中断调用被称为系统调用。实现系统调用的文件包括 system_call.s、fork.c、signal.c、sys.c 和 exit.c 文件，涉及时钟中断、出错停机、进程调度等系统调用。

通常名称以'sys_' 开头的系统调用函数都是相应系统调用需要调用的处理函数，以汇编语言实现或 C 语言实现。而名称以'do_' 开头的函数，可能是系统调用处理过程中通用的函数，也可能是某个系统调用专用的。

练习 exp6/kernel/signal.c 文件的 do_signal 函数是系统调用中断处理程序中的信号处理程序，它将信号的处理句柄插入到用户程序堆栈中，并在系统调用结束返回后立刻执行信号句柄程序，然后继续执行用户程序。

程序将用户调用系统调用的代码指针 eip 指向信号处理句柄时，使用的语句是"*(&eip) = sa_handler;"。

这条语句难道不等价于"eip = sa_handler;"？请尝试解释原因，注意函数内的变量声明。

7.4 键盘驱动

exp6/kernel/chr_drv/kb.S 是一个键盘驱动程序，主要包括键盘中断处理程序。kb.S 的 key_table 标号后的代码是一张子程序地址跳转表。当取得扫描码后就根据此表调用相应的扫描码处理子程序。

例如按下 F1-F12 按键后，kb.S 的 func 标号后的汇编指令将被执行，如下所示。

```
pushl %eax
pushl %ecx
pushl %edx
call_show_stat
pop %edx
pop %ecx
pop %eax
...
```

可知，这段汇编调用了显示各任务状态的函数 (exp7/kernel/sched.c)。

练习 参照 Linux v0.11 中已有的系统调用，尝试添加一个系统调用 sys_ver。结合键盘中断，使用户在按下 F12 时，系统屏幕打印出“NEUOS exp6”，或是你编写的具有其他功能的函数。

7.5 fork 系统调用

Linux 所有进程都是进程 0 的子进程。fork() 系统调用用于创建子进程。exp6/kernel/fork.c 是 exp6/kernel/system_call.s 的辅助处理函数集，给出了 sys_fork() 系统调用使用的两个 C 语言函数 find_empty_process() 和 copy_process()。包括进程内存区域验证与内存分配函数 verify_area()。

练习 在 exp6/kernel/fork.c 中，copy_process 的参数有 17 个，其中一个参数并没有在函数体中被使用，它对应了**堆栈**中的什么内容？请简要说明原因。

你需要熟悉函数调用时堆栈是如何被使用的，该函数由 system_call.s 中的 system_call 标号后的汇编指令调用。