School of Data and Computer Science, Sun Yat-sen University

《操作系统实验》

实验五：系统调用的实现

17341190 叶盛源

2019 年 4 月 20 日

Contents

操作系统实验报告

1 实验目的:

操作系统除了执行用户的程序，还有义务为用户程序开发提供一些常用的服务,本次实验的目的就是开发一些系统调用的程序供用户程序使用。

2 实验要求:

使用软中断实现系统服务

3 实验方案:

3.1 实验环境：

1) 实验运行环境：Windows10
2) 虚拟机软件：VMware Function和DOSBox
3) TCC+Tasm+TLink 混合编译链接
4) NASM 编译器

3.2 实验工具：

1) 汇编语言：NASM、TASM
2) 文本编辑器：VScode、notepad++
3) 软盘操作工具：WinHex

3.3 实验原理和思想：

操作系统除了执行用户的程序，还有义务为用户程序开发提供一些常用的服务，高级语言中，可以使用系统调用，实现软件重用的效果操作系统提供的服务可以用多种方式供用户程序使用。

1) 系统调用介绍：

   i) 子程序库静态链接：采用子程序调用方式，如汇编语言中用call指令调用操作系统提供服务的子程序，静态链接到用户程序代码中，这种方式

优点是程序执行快，最大缺点是用户程序内在和外存空间占用多，子程序库管理维护工作复杂。

ii) 内核子程序软中断调用：采用软中断方式，如汇编语言中用int指令调用操作系统提供服务的子程序，系统服务的子程序在内核，这种方式的优点是服务由系统提供，程序效率较高，且被所有用户程序代码共享，有利于节省内存，最大缺点是需要防止内核再入或内核设计为可再入，且用户程序陷入内核和内核返回用户程序的开销较大。

iii) 子程序库动态链接：采用动态链接技术，操作系统在运行时响应子程序调用，加载相应的子服务程序并链接致用户地址空间，这种方式优点是可由多方提供服务程序，服务更内容丰富，增加和变更服务方便，最大缺点是链接耗时多，程序响应变慢，实现复杂。

2) 修改中断的思想

i) 中断向量表：

X86计算机在启动的时候，会在内存的低位区（地址为0 1023[3FFH]，1KB）创建含256个中断向量的表IVT （每个向量[地址]占4个字节，格式为：16位段值:16位偏移值）如下图figure1。CPU根据指令的中断号找到中断向量表的位置，然后取出表中中断服务程序的入口CS:IP并跳转。在中断服务子程序结束后，我们需要使用指令iret来返回主程序。

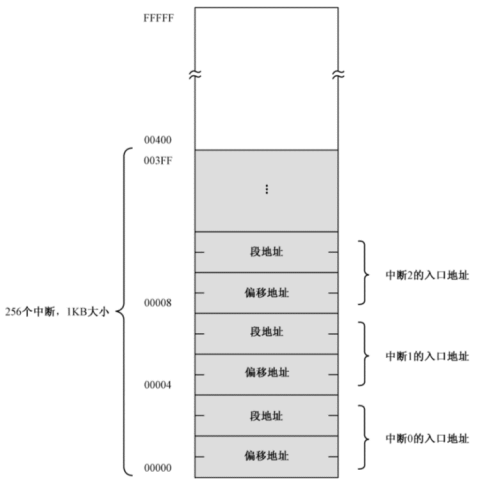

Figure 1: 中断向量表

ii) 中断开启和屏蔽：

当我们进入中断程序的时候，为了防止其他中断再进入，我们可以选择屏蔽中断。使用指令 CLI 就可以修改处理器状态字中的中断相关的对应位置，实现中断屏蔽。如果我们要开启中断的时候就可以使用 STI 指令。当我们编写的硬件中断程序结束的时候，我们需要给8258芯片发送EOI信号来将ISR对应位置清零来结束中断服务程序。

3) 软中断实现系统服务：

i) 因为操作系统要提供的服务更多，服务子程序数量太多，但中断向量有限，因此，实际做法是专门指定一个中断号对应服务处理程序总入口，然后再将服务程序所有服务用功能号区分，并作为一个参数从用户中传递过来，服务程序再进行分支，进入相应的功能实现子程序。

ii) 这种方案至少要求向用户公开一个中断号和参数表，即所谓的系统调用手册，供用户使用。

iii) 如果用户所用的开发语言是汇编语言，可以直接使用软中断调用

iv) 如果使用高级语言，则要用库过程封装调用的参数传递和软中断等指令汇编代替

v) 规定系统调用服务的中断号是21h。

3.4 实验过程和思想：

1) 编写21h号中断服务程序

在内核中新增int_21h号程序段，并将原来编写的内核函数中间的程序段搬进新增的中断程序段中。要注意的是，搬进中断程序后因为变成了中断调用，返回的指令要改成使用iret而不是ret。而原来的子过程就要改为一个简单的中断程序调用，使用指令 int 21h 。下图举例了将 printchar 子过程移入中断程序段中，其他以此类推。

因为内核中的系统服务程序比较多，我们在int_21过程中可以使用功能号来实现系统服务。功能号一般放在寄存器 ah 中，所以在子过程前我们需要用cmp函数搭配jz函数做一个类似 switch case 语句段。但会发现因为跳转距离过长，导致jz无法跳转，所以要改为使用jnz搭配jmp来实现跳转更长的距离到目标程序段。下图展示了示例代码。

```
printChar:
    push bp
    mov bp,sp
    mov al,[bp+10];char\ip\bp\ip\cs\psw  bp 才能寻址
    mov bl,0
    mov ah,0eh
    int 10h
    mov sp,bp
    pop bp
    iret
```

Figure 2: 原来的printchar函数

```
printChar:
    push bp
    mov bp,sp
    mov al,[bp+10];char\ip\bp\ip\cs\psw  bp 才能寻址
    mov bl,0
    mov ah,0eh
    int 10h
    mov sp,bp
    pop bp
    iret
```

Figure 3: 移入中断的printchar程序段

```
int_21h:
    cmp ah,0
    jz printChar
    cmp ah,1
    jnz next1
    jmp Readchar
next1:
    cmp ah,2
    jnz next2
    jmp clear
next2:
    cmp ah,3
    jnz next3
    jmp RunProm
next3:

next4:
    cmp ah,5
    jnz next5
    jmp loadListFromDisk
next5:
    cmp ah,6
    jnz next6
    jmp loadbatchFromDisk
next6:
    cmp ah,7
    jnz next7
    jmp readbatch
next7:
    cmp ah,8
    jnz quit
    jmp readList
quit:
```

Figure 4: 根据功能号跳转

2) 修改中断向量表

所有的中断子程序都已经写完了，现在们需要修改中断向量表来让我们可以通过中断进入我们的子程序中，我在 monitor.asm 文件中在调用 cmain 函数前使用语句修改了向量表。我们通过中断向量号计算出内存的位置，再依次插入IP和CS的值完成修改，如下图：

```
mov word ptr es:[33*4], offset int_21h
mov word ptr es:[33*4+2], cs
```

Figure 5: 修改中断向量表

3) 用软盘启动裸机:

将内核用tcc+tasm+tlink编译链接成.com文件，再把5个用户程序、list、batch和loader用nasm编译形成bin文件，加入到1.44MB软盘的合适扇区中并运行程序，虚拟机运行如下图：

Figure 6: 开机界面

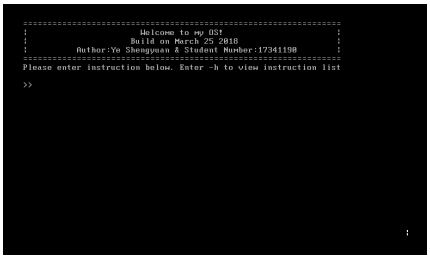

Figure 7: 操作系统界面

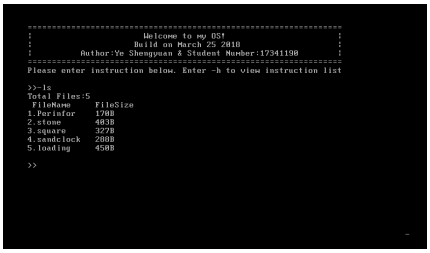

Figure 8: 文件列表界面

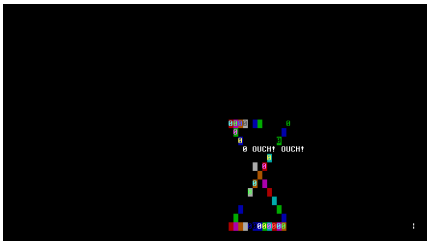

Figure 9: 用中断开启的用户程序

4 实验心得和总结：

本次实验可以看做我们之前完成的简单内核的一个优化，也是和接下来实验的一个过渡，本次实验比起其他的时间过程比较简单，花费时间也不是很长，但也收获颇丰。

我们之前实现的操作系统高级语言调用内核函数的时候是直接调用汇编实现的子过程。但在真正的操作系统中，用户程序是不能直接触碰内核的，所以不可能就这样直接去随意的调用。不过操作系统也要实现一些简单的服务程序给用户程序去使用，所以就会在 `int 21h` 这个中断服务号里实现一些服务程序并提供接口给用户程序去调用，内核并不全部开放让用户程序直接去取来用，这个在接下来的实验中是十分重要的，所以我们需要做这个修改，既是对我们之前实验的一个完善，也是对接下来实验的一个准备。

但其实在这次的实验中，并不是想象中那么顺利，我还是遇到了一些问题。例如一开始忘记了要在中断服务程序返回的时候使用iret，仍然使用ret，所以导致了我程序一直返回出错。后面在移动一个读list文件的函数进入中断程序的时候，也遇到了问题，就是一段一模一样的代码，在外面能正常运行，但在中断中就一直出错。后来才发现是寄存器ah没有清零，之前因为没有使用过ah所以没有出错。但这次因为要往ah中放功能号，ah的值不再是0，所以就出现了

错误，在我的更改之后终于完成了实验。

在做实验前，就听很多同学说，这个实验比较简单，很多同学就20分钟完成了，但亲自上手的时候，缺花了不少的时间，原因就是我之前的代码不规范也不完善，导致了简单的移动反而出现了很多的问题，经过这次实验，优化了我的代码结构和规范了代码，为后面更好的完成实验打下了基础。

总而言之，这次实验还是比较顺利的完成了，希望在接下来的时间里还能继续完成更多的实验，学习到更多的知识!

5 参考文献:

1) nasm手册

2) https://blog.csdn.net/longintchar/article/details/79511747

3) https://blog.csdn.net/cielozhang/article/details/6171783/

4) https://blog.csdn.net/lulipeng_cpp/article/details/8161982

5) https://www.cnblogs.com/alwaysking/p/7789282.html

6) https://blog.csdn.net/cielozhang/article/details/6171783

7) https://blog.csdn.net/lulipeng_cpp/article/details/8161982 https://stackoverflow.com/questions/12882342/override-default-int-9h