

# 实验5：实现系统调用

院系	专业	年级	姓名
数据科学与计算机学院	人工智能与大数据	2018级	陈琮昊

## 一、实验目的：

- 1、学习掌握PC系统的软中断指令
- 2、掌握操作系统内核为用户提供服务的系统调用程序设计方法
- 3、掌握C语言的库设计方法
- 4、掌握用户程序请求系统服务的方法

## 二、实验要求：

- 1、了解PC系统的软中断指令的原理
- 2、掌握x86汇编语言软中断的响应处理编程方法
- 3、扩展实验四的的内核程序，增加输入输出服务的系统调用。
- 4、C语言的库设计，实现putch()、getch()、printf()、scanf()等基本输入输出库过程。
- 5、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

## 三、实验方案：

### 1.运行环境：

该实验在 windows+Linux 下，选用 gcc+NASM+ld 的组合运行。

所需软件：vmware workstation 15.5 Pro (windows)、WinHex 19.8 (windows)、WSL Ubuntu 18.04 LTS、GCC 7.5.0 (Linux)、NASM 2.13.02 (Linux)、GNU ld 2.30 (Linux)

虚拟机 vmware 用来运行程序，WSL Ubuntu 18.04 LTS 是以软件形式在 windows 下运行的 Linux 子系统，NASM 为汇编语言编译器，WinHex 用来将文件用16进制显示，GCC 为C语言编译器，GNU ld 用来链接代码。

### 2.实验流程：

- 1.配备好相关的软硬件
- 2.编写代码以实现相关功能
- 3.得到所需镜像文件
- 4.导入虚拟机运行并观察结果

其中编写代码在 windows 的文本编辑器下进行；编译、链接则在 wsl 下进行；最终在虚拟机观察运行结果则是在 windows 下的 vmware。

### 3.程序分析：

程序结构如下：

磁头	扇区	大小	内容
0	1	512B	引导程序
0	2	512B	存放用户程序信息
0	3-18	8KB	内核
1	1-2	1KB	用户程序1
1	3-4	1KB	用户程序2
1	5-6	1KB	用户程序3
1	7-8	1KB	用户程序4
1	9	1KB	系统调用

文件目录：

主题	所包含文件
内核	osstarter.asm、liba.asm、libc.c (包含 stringio.h)
用户程序	stone1u.asm、stone1d.asm、stoneru.asm、stonerd.asm
引导程序	bootloader.asm
相关功能	usrproginfo.asm、hotwheel.asm
宏定义	macro.asm
系统调用	syscall_test.asm、systema.asm、systemc.c

接下来对每个模块的重点进行分析：（与实验4相同的模块不再进行分析）

#### (1) ah=00h的系统调用：

本系统调用的功能是在屏幕某处显示 int 00h。

利用 BIOS 的 int 10h 来实现。在 systema.asm中封装 sys\_show 函数：

```
sys_show:
    pusha                ; 保护现场
    push ds
    push es
```

```

mov ax, cs      ; 置其他段寄存器值与CS相同
mov ds, ax      ; 数据段
mov bp, ouch_str ; BP=当前串的偏移地址
mov ax, ds      ; ES:BP = 串地址
mov es, ax      ; 置ES=DS
mov cx, 7       ; CX = 串长
mov ax, 1301h   ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
mov bx, 0038h   ; 页号为0(BH = 0) 黑底白字(BL = 07h)
mov dh, 12      ; 行号
mov dl, 38      ; 列号
int 10h         ; BIOS的10h功能: 显示一行字符
pop es
pop ds
popa            ; 恢复现场
ret
ouch_str db 'int 00h'

```

## (2) ah=01h的系统调用:

本系统调用的功能是将一个字符串中的小写字母变为大写字母。

用 C 语言实现此功能比较方便。在 `systemc.c` 中实现 `toupper` 函数:

```

/* 将字符串中的小写字母转换为大写 */
void toupper(char* str) {
    int i=0;
    while(str[i]) {
        if (str[i] >= 'a' && str[i] <= 'z')
            str[i] = str[i] - 'a' + 'A';
        i++;
    }
}

```

在 `systema.asm` 中封装:

```

[extern toupper]
sys_toUpper:
    push es      ; 传递参数
    push dx      ; 传递参数
    call dword toupper
    pop dx       ; 丢弃参数
    pop es       ; 丢弃参数
    ret

```

## (3) ah=02h的系统调用:

根据“实验要求”，本系统调用的功能是将一个字符串中的大写字母变为小写字母。

与上一条基本类似，C 代码:

```

/* 将字符串中的大写字母转换为小写 */
void tolower(char* str) {
    int i=0;
    while(str[i]) {
        if (str[i] >= 'A' && str[i] <= 'Z')
            str[i] = str[i] - 'A' + 'a';
        i++;
    }
}

```

在 systema.asm 中封装:

```

[extern tolower]
sys_toLower:
    push es          ; 传递参数
    push dx          ; 传递参数
    call dword tolower
    pop dx           ; 丢弃参数
    pop es           ; 丢弃参数
    ret

```

#### (4) 编写系统调用入口向量程序:

需要编写一段代码来根据功能号去执行不同的系统调用。在 liba.asm 中增加一个函数:

```

[extern sys_show]
[extern sys_toUpper]
[extern sys_toLower]
syscaller:
    push ds
    push si          ; 用si作为内部临时寄存器
    mov si, cs
    mov ds, si       ; ds = cs
    mov si, ax
    shr si, 8         ; si = 功能号
    add si, si        ; si = 2 * 功能号
    call [sys_table+si] ; 系统调用函数
    pop si
    pop ds
    iret              ; int 21h中断返回
sys_table:            ; 存放功能号与系统调用函数映射的表
    dw sys_showOuch, sys_toUpper, sys_toLower

```

以上函数通过 ah 的值在 sys\_table 表中找到对应位置的项去执行。

还要提前把 syscaller 放入 21h 号中断向量。在 osstarter.asm 中添加一行宏指令完成此操作:

(宏实现在 macro.asm 文件中)

```
WRITE_INT_VECTOR 21h, syscaller
```

## (5) 编写测试用的用户程序:

编写用户程序 `syscall_test`，来测试上面实现的系统调用。该用户程序的期望效果如下所述：

- 首先显示一行提示信息，表示已经进入 `syscall_test` 程序，并提示用户按下 `ENTER` 以开始测试系统调用。在以下步骤中，每测试完一个功能后，用户都需要按下 `ENTER` 继续测试下一个功能；
- 测试 `ah=00h` 功能，期望在屏幕中间显示出 `int 00h`。

```
mov ah, 00h                ; 系统调用功能号ah=00h, 显示00h
int 21h
mov ah, 0
int 16h
cmp al, 101                ; 按下e
je QuitUsrProg             ; 直接退出
```

- 后面几个功能类似，此处不再赘述。

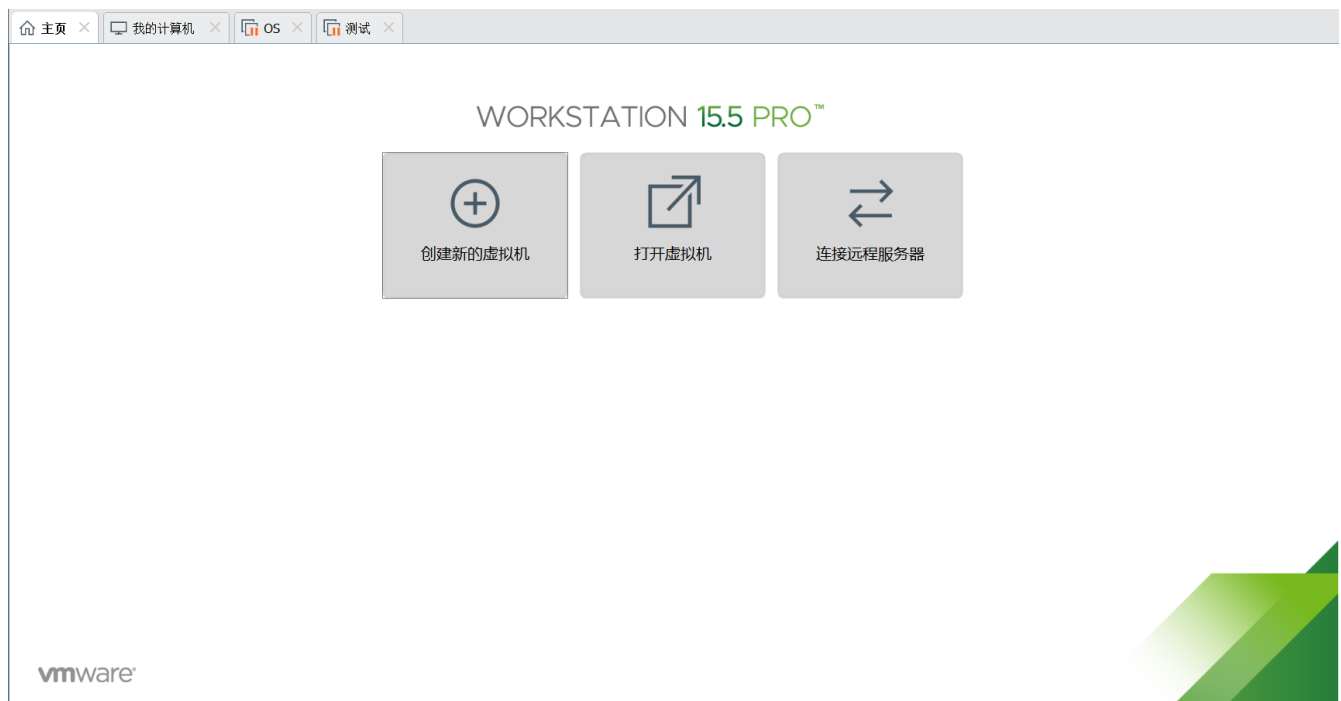
## 四、实验过程:

### 1.软件的下载与安装:

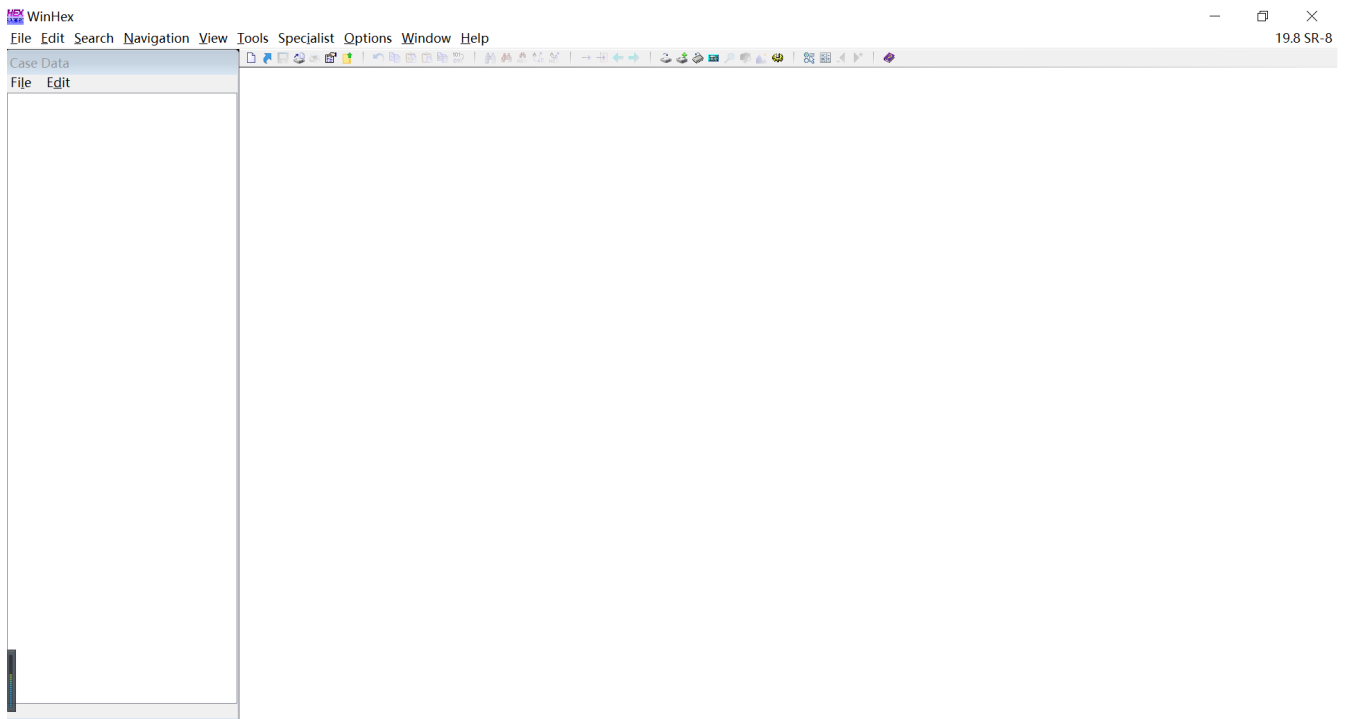
(1) 在官网下载 VMware Workstation 15.5 Pro:

<https://www.vmware.com/cn/products/workstation-pro/workstation-pro-evaluation.html>

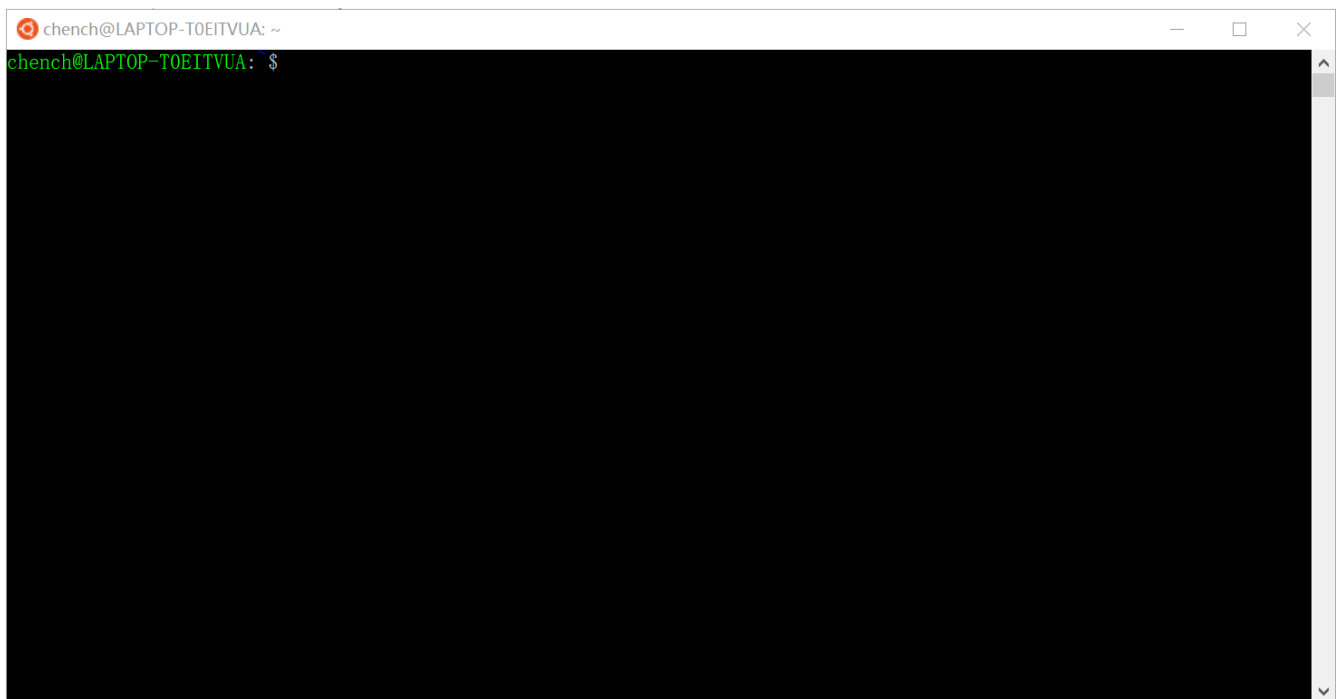
安装该软件后初始界面如下（已经输入密钥并初步设置后）：



(2) 下载 WinHex 19.8 压缩包，解压后找到 `winhex.exe`，打开并解锁，界面如下：



(3) 配置 Linux 环境, 使用 windows Subsystem for Linux (Ubuntu 18.04 LTS) :



然后输入相关指令安装 gcc、NASM、GNU ld, 至此本次实验所需软件已配备完成。
















## 2.Windows下编写代码：

代码关键部分已在 程序分析 一栏中介绍, 具体代码内容见 src 文件夹。

## 3.配置相关文件：

首先, 进入 WSL 的目录, 将需要在 Linux 下编译的代码放在 WSL 目录下：

> wsl\$ > Ubuntu-18.04 > home > chench >

名称	修改日期	类型	大小
 kernel.bin	2020/6/19 17:27	BIN 文件	7 KB
 liba.o	2020/6/19 17:27	O 文件	3 KB
 systema.o	2020/6/19 17:27	O 文件	1 KB
 systema.asm	2020/6/19 17:27	ASM 文件	2 KB
 liba.asm	2020/6/19 17:25	ASM 文件	10 KB
 systemc.o	2020/6/19 16:30	O 文件	2 KB
 libc.o	2020/6/19 16:30	O 文件	9 KB
 hotwheel.o	2020/6/19 16:30	O 文件	1 KB
 osstarter.o	2020/6/19 16:30	O 文件	1 KB
 macro.asm	2020/6/19 16:23	ASM 文件	3 KB
 libc.c	2020/6/19 15:51	C Source File	7 KB
 osstarter.asm	2020/6/19 15:47	ASM 文件	1 KB
 stringio.h	2020/6/19 15:46	C Header File	4 KB
 systemc.c	2020/6/19 15:46	C Source File	2 KB
 hotwheel.asm	2020/6/19 15:45	ASM 文件	2 KB

其次，进行编译链接：（借用上次实验且无需改动的程序就未再次进行编译）

```
chench@LAPTOP-TOEITVUA:~$ nasm -f elf32 hotwheel.asm -o hotwheel.p
chench@LAPTOP-TOEITVUA:~$ nasm -f elf32 hotwheel.asm -o hotwheel.o
chench@LAPTOP-TOEITVUA:~$ nasm -f elf32 osstarter.asm -o osstarter.o
chench@LAPTOP-TOEITVUA:~$ nasm -f elf32 liba.asm -o liba.o
chench@LAPTOP-TOEITVUA:~$ nasm -f elf32 systema.asm -o systema.o
chench@LAPTOP-TOEITVUA:~$ gcc -march=i386 -m16 -mpreferred-stack-boundary=2 -ffreestanding -fno-PIE -masm=intel -c systemc.c -o systemc.o
chench@LAPTOP-TOEITVUA:~$ gcc -march=i386 -m16 -mpreferred-stack-boundary=2 -ffreestanding -fno-PIE -masm=intel -c libc.c -o libc.o
chench@LAPTOP-TOEITVUA:~$ ld -m elf_i386 -N --oformat binary -Ttext 0x8000 osstarter.o liba.o libc.o systema.o systemc.o hotwheel.o -o kernel.bin
```

编译完成后可以在该目录下找到上述所有 .bin 文件。

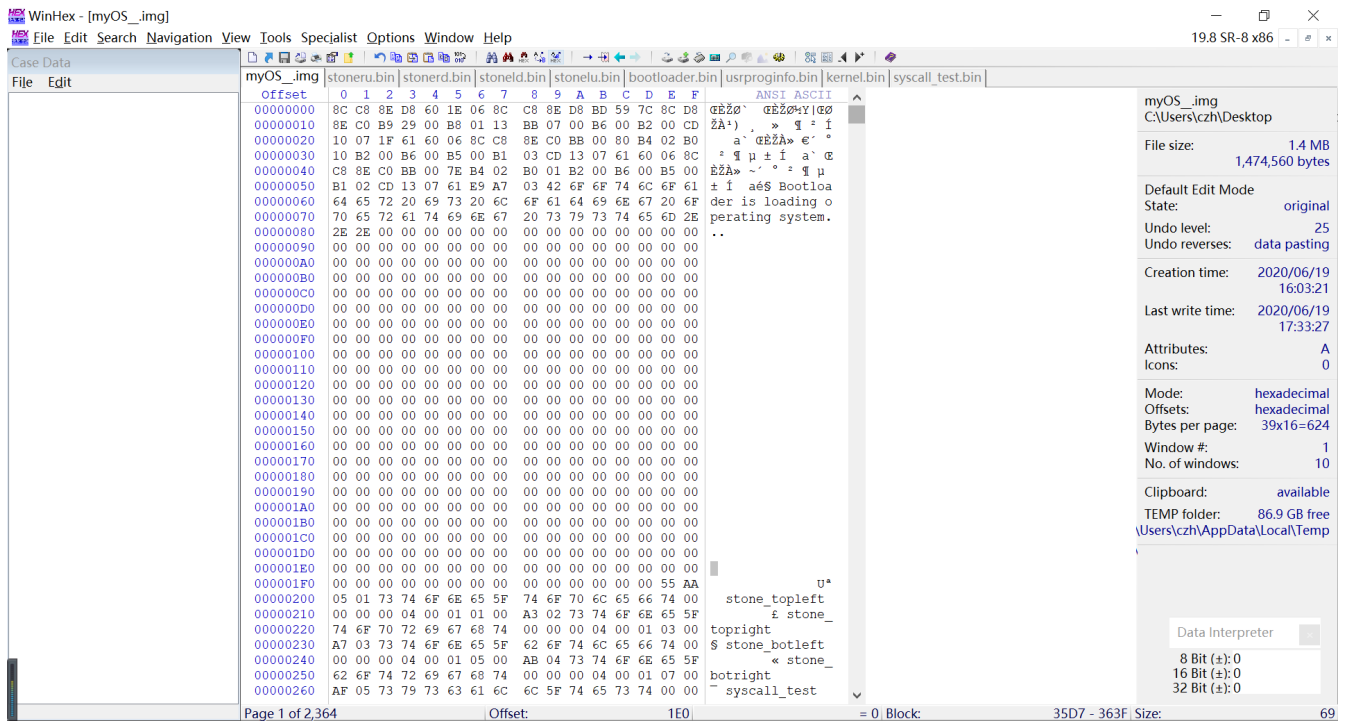
最后，便是得到本次实验所需的镜像文件：

将实验4中生成的镜像文件 myOS\_.img 复制一份并在 winHex 内打开，需要改动的地方将新生成的 bin 文件进行复制粘贴，将原文件的内容进行覆盖，命名为 myOS\_.img。

打开制作软盘所需的所有文件（共8个），然后将这8个文件按照如下顺序复制粘贴至新建的文件中：

```
bootloader.bin->usrproginfo.bin->kernel.bin->stonelu.bin->stoneru.bin->stone1d.bin->stonerd.bin->syscall_test.bin
```

程序分析 部分已经列出了每个文件的扇区、大小等相关信息。因此，复制粘贴时需要注意 offset，如 usrproginfo.bin 的偏移为 0x0200，kernel.bin 的偏移为 0x0400，第一个用户程序的偏移为 0x2400 .....



至此，便得到了命名为 myOS\_.img 的镜像文件。

## 4.导入虚拟机：

打开虚拟机，选择 创建新的虚拟机 一项，然后进行如下配置：

### 安装客户机操作系统

虚拟机如同物理机，需要操作系统。您将如何安装客户机操作系统？

☒ 稍后安装操作系统(S)。

创建的虚拟机将包含一个空白硬盘。

帮助

< 上一步(B)

下一步(N) >

取消



选择客户机操作系统

此虚拟机中将安装哪种操作系统？

客户机操作系统

☐ Microsoft Windows(W)

☐ Linux(L)

☐ VMware ESX(X)

☒ 其他(O)

版本(V)

其他

▼

帮助

< 上一步(B)

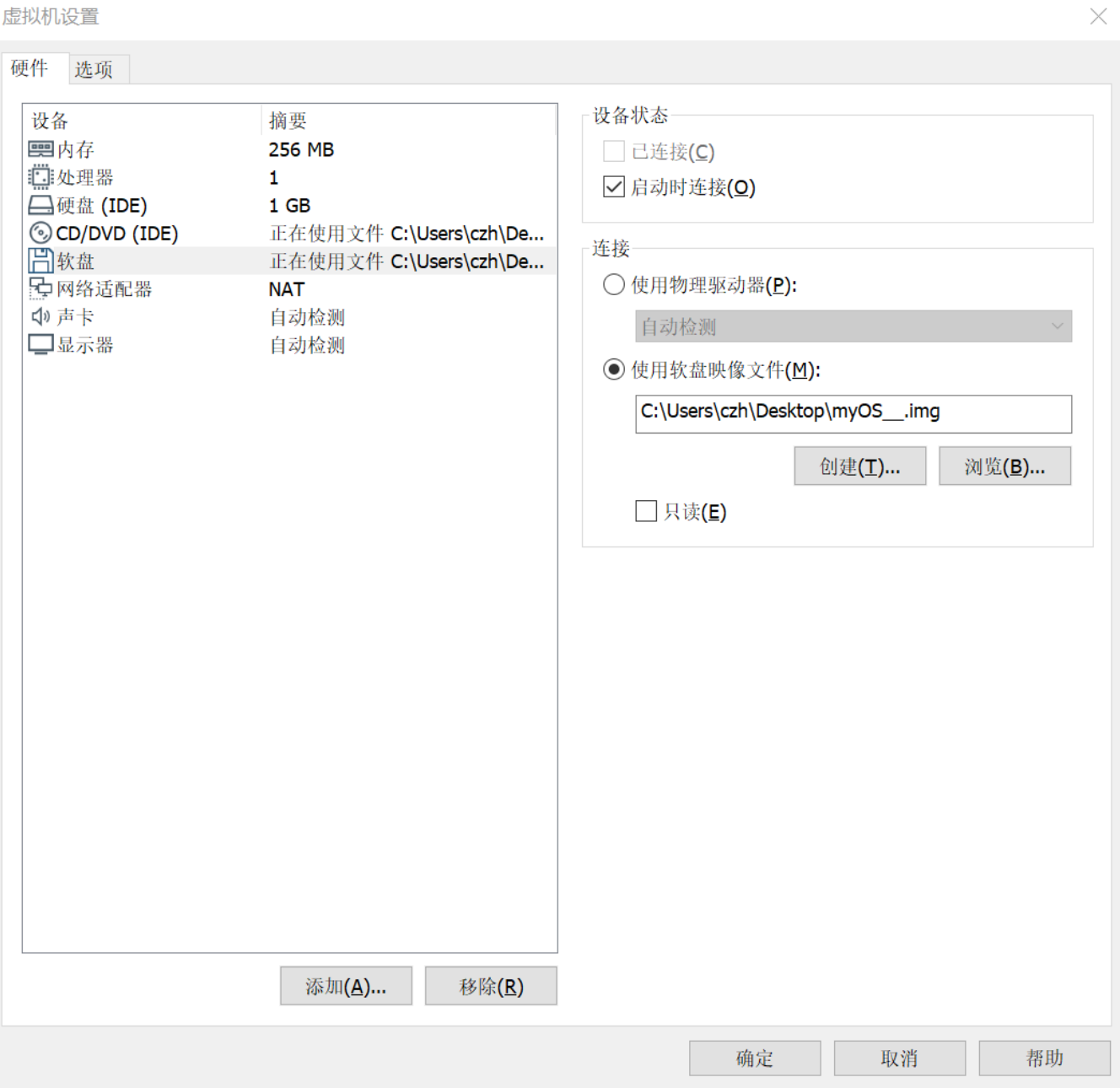
下一步(N) >

取消

于是所使用的虚拟机也已配置完成。（后续步骤中选择磁盘容量可视情况而定，其他步骤则不赘述了。）

在上一步中，已经通过 WinHex 得到了最终的 myOS\_\_.img 镜像文件，接下来就是将 myOS\_\_.img 文件导入虚拟机，步骤如下：

打开已创建好的虚拟机，然后选择 编辑虚拟机设置，然后添加 软盘，在 连接 一栏选择 使用软盘映像文件，将 myOS\_\_.img 导入， 确定 即可。



5.运行显示:

添加软盘完成后，打开虚拟机，从虚拟软盘启动后即可看到如下结果：

(1) 初始界面：

```
ChenConghao 18340013
```

```
myOS has been loaded successfully. Press ENTER to start!
```

显示了相关信息，可以看到右下角有"风火轮"。

(2) 按 `ENTER` 进入后：

```
Welcome to myOS. These commands can be execute.
```

```
help - show information about myOS
clear - clear screen
list - show the information about user program
run - run user program in sequence. e.g. 'run 3 2 1'
date - show year month day hour minute second
hotwheel - turn on/off the hotwheel
poweroff - force close the machine
```

```
myOS >> _
```

可以看到显示了提示信息。

(3) 执行 `help` 操作：

```
Welcome to myOS. These commands can be execute.

    help - show information about myOS
    clear - clear screen
    list - show the information about user program
    run - run user program in sequence. e.g. 'run 3 2 1'
    date - show year month day hour minute second
    hotwheel - turn on/off the hotwheel
    poweroff - force close the machine
myOS >> help
Welcome to myOS. These commands can be execute.

    help - show information about myOS
    clear - clear screen
    list - show the information about user program
    run - run user program in sequence. e.g. 'run 3 2 1'
    date - show year month day hour minute second
    hotwheel - turn on/off the hotwheel
    poweroff - force close the machine
myOS >>
```

显示了可执行的指令及意义。

(4) 执行 `clear` 操作:

```
myOS >> _
```

可以看到清屏后只留下了命令行。

(5) 执行 `list` 操作:

```

Welcome to myOS. These commands can be execute.

    help - show information about myOS
    clear - clear screen
    list - show the information about user program
    run - run user program in sequence. e.g. 'run 3 2 1'
    date - show year month day hour minute second
    hotwheel - turn on/off the hotwheel
    poweroff - force close the machine
myOS >> list
You can use 'run <PID>' to run a user programme.
PID -      Name      -  Size  -  Addr - Cylinder - Head - Sector
1 -  stone_topleft   -  1024 -  A300 -      0 -      1 -      1
2 -  stone_topright  -  1024 -  A700 -      0 -      1 -      3
3 -  stone_botleft   -  1024 -  AB00 -      0 -      1 -      5
4 -  stone_botright  -  1024 -  AF00 -      0 -      1 -      7
5 -  syscall_test    -  1024 -  B300 -      0 -      1 -      9
myOS >> _

```

显示了4个用户程序以及测试系统调用程序的信息。

(6) **run 操作**：在 run 操作中，以下几种命令合法：**run 1**（只执行一个用户程序）、**run 4 3 2 1**（按一定顺序执行多个用户程序）、**run 1 1 1**（连续执行相同的用户程序），且run后面的数字只能是1-4（见后面可知**run 5是执行系统调用**）。在执行多个程序过程中，中途可按 e 退出进入下一程序。所有程序执行完毕后会返回到该界面：

```

All programmes have been executed successfully as you wish.
myOS >>

```

注：该功能同实验4完全一致，就不再赘述了。

(7) 执行系统调用：键入 `run 5` 即可进入系统调用界面，进入后通过按 `ENTER` 来执行不同的调用：首先是在某位置显示 `int 00h` 字符，然后会给出一个字符串，先是将该字符串的所有字母均转为大写，然后将该字符串的所有字母均转为小写，再按一次 `ENTER` 就退出了系统调用。当然，在中间过程按 `e` 可直接退出返回命令行。

```
Welcome to syscall test program, please press ENTER to start and e to exit

AbCdEfGhIjKlMn
ABCDEFGHijklmn
abcdefghijklmnopqrstuvwxyz_

int 00h
```

(8) 执行 `hotwheel` 操作：默认情况下会在右下角显示"风火轮"，执行一次 `hotwheel` 则关闭"风火轮"，再执行一次 `hotwheel` 则又重新开启。该功能同实验4完全一致，此处不再赘述。

(9) 执行 `date` 指令：显示当前时间。

```
All programmes have been executed successfully as you wish.
myOS >> date
2020-6-19 18:9:58
myOS >>
```

(10) 执行 `poweroff` 操作：直接关闭虚拟机。



(11) 如果输入不正确的指令（如 `aaa`），则会显示 `command not found`：

```
All programmes have been executed successfully as you wish.
myOS >> date
2020-6-19 18:9:58
myOS >> help
Welcome to myOS. These commands can be execute.

help - show information about myOS
clear - clear screen
list - show the information about user program
run - run user program in sequence. e.g. 'run 3 2 1'
date - show year month day hour minute second
hotwheel - turn on/off the hotwheel
poweroff - force close the machine
myOS >> aaa
aaa: command not found
myOS >>
```

## 五、实验总结：

本次实验个人感觉有点困难，其实难点就是在 `系统调用`，因为本次实验同上次实验相比除了增加 `系统调用` 这一功能外，其余内容一模一样，几乎不用改动。`系统调用`大致就是这样一个流程：硬件接收到中断信号，立刻保存现场，并查找中断向量表，进入系统调用；执行系统调用处理程序；恢复现场，返回用户程序。搞清楚这样一个过程后就比较容易上手了。本次实验的最终呈现效果同上次实验相比比较简洁，没有增加彩色字符等不是很实用的功能，毕竟重点不在这里。在网上也参考了一些相关资料，也参考了“大佬”的一些想法，可以说让我受益匪浅。维持 `栈平衡` 是非常重要的点，如果这一点搞不清楚，代码调试起来会没有头绪；而且在群里也可以看到，大家讨论较多的就是栈的使用

的相关问题，因此更提醒我要注意这一点。堆栈入栈时从高地址向低地址增长，且一般使用 `bp` 寄存器来跟踪参数。调用函数时，需要先用 `push` 指令将参数依次压栈，右边的参数先入栈。参数都按顺序入栈以后，才能使用 `call` 指令进行子函数调用。子函数返回后，`caller` 要负责将栈中的参数丢弃掉，这称为 栈平衡，其意义是保证函数调用前后 `sp` 是一致的。常用的栈平衡是 外平衡，即由函数外部保持栈平衡，因此在函数调用之后要使用 `pop` 命令将参数丢弃掉。还需要注意的地方就是随着实验的进行，内核占用的空间越来越大，因此需要注意空间的分配！一个 1.44MB 的软盘由80个磁道、18个扇区构成，而且有2个柱面。首先使用的是0柱面、0磁道的扇区，扇区编号从1到18。再往后，是0柱面、1 磁道，扇区号又是从1到18.....依此类推。

## 六、参考文献：

---

李忠，王晓波，余洁．《x86汇编语言：从实模式到保护模式》．电子工业出版社，2012．

王爽．《汇编语言（第3版）》．

凌应标．04实验课.pptx

<https://www.cnblogs.com/secoding/p/11414172.html>

[https://blog.csdn.net/qq\\_32623363/article/details/87858888](https://blog.csdn.net/qq_32623363/article/details/87858888)