

实验4：具有中断处理的内核

| 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、PC系统的中断机制和原理
- 2、理解操作系统内核对异步事件的处理方法
- 3、掌握中断处理编程的方法
- 4、掌握内核中断处理代码组织的设计方法
- 5、了解查询式I/O控制方式的编程方法

二、实验要求：

- 1、知道PC系统的中断硬件系统的原理
- 2、掌握 x86 汇编语言对时钟中断的响应处理编程方法
- 3、重写和扩展实验三的的内核程序，增加时钟中断的响应处理和键盘中断响应
- 4、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.亮点工作：

1.彩色字符：

在上次实验中非用户程序打印出来的只有白色字符，本次实验中增加了彩色字符。

本次实验增加了一个新的函数 putchar_c，即显示一个可选颜色的字符，并移动光标到下一位置。

该函数由 C 语言调用，函数原型如下：

```
extern void putchar_c(char c, uint8_t color);
```

该函数主要实现了两件事：

1. 使用 ah=03h 的 int 10h 以获取当前光标位置，并将行列保存在 dh 和 dl 中；
2. 使用 ah=13h 的 int 10h 显示字符串。令 cx=1 从而实现只打印一个字符，字符的地址从栈中取得，字符的颜色属性也从栈中取得。这样就可以使光标自动变换到下一个字符的位置。

汇编定义如下：

```
putchar_c:
    pusha
    push ds
    push es
    mov bx, 0                ; 页号=0
    mov ah, 03h              ; 功能号：获取光标位置
    int 10h                  ; dh=行, dl=列
    mov ax, cs
    mov ds, ax               ; ds = cs
    mov es, ax               ; es = cs
    mov bp, sp
    add bp, 20+4              ; 参数地址, es:bp指向要显示的字符
    mov cx, 1                 ; 显示1个字符
    mov ax, 1301h             ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
    mov bh, 0                 ; 页号
    mov bl, [bp+4]            ; 颜色属性
    int 10h                   ; 显示字符串 (1个字符)
    pop es
    pop ds
    popa
    retf
```

有了上述函数后，打印彩色字符串就很简单了，用C语言的几行语句就可实现：

```
void print_c(const char* str, uint8_t color) { //打印彩色字符串函数
    for(int i = 0, len = strlen(str); i < len; i++) {
        putchar_c(str[i], color);
    }
}
```

2."风火轮"开关:

该命令通过汇编语言实现于 ka.asm 中。原理如下: 判断 08h 号中断向量指向的是不是风火轮函数 Timer 的地址, 如果是, 则将其恢复为初始的中断处理程序; 如果不是, 则将其替换为风火轮中断处理程序, 即 Timer。

为了增加这个功能, 需要先将原本的 08h 号中断向量移动到 38h 去。将 kernel.asm 中的相关代码改为:

```
MOVE_INT_VECTOR 08h, 38h
WRITE_INT_VECTOR 08h, Timer ; 装填时钟中断向量表
```

切换开关状态的代码如下:

```
switchHotwheel:                ; 函数: 打开或关闭风火轮
    push es
    mov ax, 0
    mov es, ax
    mov ax, [es:08h*4]          ; ax=08h号中断处理程序的偏移地址
    cmp ax, Timer               ; 检查08h号中断处理程序是否是风火轮
    je turnoff                  ; 如果是, 则关闭
    WRITE_INT_VECTOR 08h, Timer ; 如果不是, 则打开
    mov ax, 1                   ; 返回1表示风火轮已打开
    jmp switchDone
turnoff:
    MOVE_INT_VECTOR 38h, 08h
    mov ax, 0B800h              ; 文本窗口显存起始地址
    mov gs, ax                  ; GS = B800h
    mov ah, 0Fh                 ; 黑色背景
    mov al, ' '                 ; 显示空格
    mov [gs:((80*24+79)*2)], ax ; 更新显存
    mov ax, 0                   ; 返回0表示风火轮已关闭
switchDone:
    pop es
    retf
```

该函数在汇编实现后通过 C 语言调用。调用该函数后屏幕右下角的风火轮就会开、关。

3.显示当前时间:

在 PC 机中有一个 CMOS RAM 芯片, 其中包含一个时钟, 当前时间就存放在这里。年、约、日、时、分、秒这 6 个信息各占 1 个字节, 以 BCD 码的形式存放在 CMOS 中, 存放单元依次为 0、2、4、7、8、9。要读取 CMOS 信息, 首先向 70h 端口写入要访问的单元地址, 然后从 71h 端口读入数据。

读取日期和时间的功能被实现为 ka.asm 中的 6 个函数, 它们的全局声明如下:

```
[global getDateYear]
[global getDateMonth]
[global getDateDay]
[global getDateHour]
[global getDateMinute]
[global getDateSecond]
```

以上六个函数定义原理相同，这里取 `getDateYear` 作为示例。该函数从 71h 端口读出单元 9 中的内容，将其存放在 `ax` 中作为返回值：

```
getDateYear:                ; 从CMOS获取当前年份
    mov al, 9
    out 70h, al
    in al, 71h
    mov ah, 0
    retf
```

这些函数供内核中的C代码调用。与之对应的C函数原型如下：

```
extern uint8_t getDateYear();
extern uint8_t getDateMonth();
extern uint8_t getDateDay();
extern uint8_t getDateHour();
extern uint8_t getDateMinute();
extern uint8_t getDateSecond();
```

需要注意的是，这些函数返回的是 `BCD` 码，要用以前学到的知识将 `BCD` 码转换为十进制：

```
uint8_t bcd2decimal(uint8_t bcd)
{
    return ((bcd & 0xF0) >> 4) * 10 + (bcd & 0x0F);
}
```

然后在C代码中要求 `date` 指令输入时显示上述信息即可。

4.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 |

文件目录：

| 主题 | 所包含文件 |
|------|---|
| 内核 | kernel.asm、ka.asm、kc.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、intouch.asm |
| 宏定义 | macro.asm |

接下来对每个模块的重点进行分析：（与实验3相同的模块不再进行分析）

（1）hotwheel.asm：

系统时钟中断号为 08h，该中断由系统每隔一个固定的时间自动调用，因此只需要将中断处理程序写入中断向量表中 08h 号的位置即可。

中断处理程序要求在屏幕右下角（第24行，第79列）的位置循环依次显示' -\|/' 这四个字符。基本思路很简单：使用一个寄存器 si 来保存当前显示的字符是这四个字符中的哪一个，然后每隔一段时间递增该寄存器，如果递增到4了就将其归零，如此循环往复。

在中断处理程序中，首先检查 count 变量是否递减为 0，如果否，则立即中断返回，不做任何操作；如果是0，则递增 si，使其指向下一个要显示的字符，并更新显存。

关键代码：

```
dec byte [count]          ; 递减计数变量
jnz EndInt                ; >0: 跳转
mov byte[count],delay     ; 重置计数变量=初值delay
mov si, hotwheel          ; 风火轮首字符地址
add si, [wheel_offset]    ; 风火轮字符偏移量
mov al, [si]              ; al=要显示的字符
mov ah, 0Ch               ; ah=黑底，淡红色
mov [gs:((80*24+79)*2)],ax ; 更新显存
```

```

inc byte[wheel_offset]      ; 递增偏移量
cmp byte[wheel_offset], 4   ; 检查偏移量是否超过3
jne EndInt                  ; 没有超过, 中断返回
mov byte[wheel_offset], 0   ; 超过3了, 重置为0

... ; 涉及键盘检测的代码已经略去

mov ax, [end_row]           ; al=end_row, ah实际上无用
mov ah, 2*80
mul ah                      ; ax=end_row * 2 * 80
add ax, [start_row]         ; ax=start_row + end_row*2*80
cmp si, ax
jne disploop                ; 范围全部显示完, 中断返回

```

该中断处理程序与内核一起编译, 并在进入内核开始时将其写入中断向量表中即可。

在内核 (kernel.asm) 中写入 08h 号中断向量:

```
WRITE_INT_VECTOR 08h, Timer ; 装填时钟中断向量表
```

(2) intouch.asm:

当按下键盘时, 系统引发 09h 号中断, 在 BIOS 提供的中断处理程序中, 首先从 60h 端口中读出按键扫描码, 然后将其存入键盘缓冲区。而键盘缓冲区中的内容可以由 int 16h 读出并清除。

为了能够在用户程序中按下键盘时能够显示"OUCH! OUCH!"字符串, 需要手动编写 int 09h 的中断处理程序, 在该程序中, 需要在屏幕特定位置显示出"OUCH! OUCH!", 还要读入按下的字符, 避免其堵塞在 60h 端口或残留在 BIOS 键盘缓冲区中。

实现思路如下: 在自己编写的中断例程中调用原来 BIOS 提供的中断处理程序。先保存原来程序的地址, 然后再把自己的程序写入中断向量表, 并在自己的程序中调用原来的程序。

接下来则是如何编写中断处理程序 IntOuch。

为了能够看清显示的"OUCH! OUCH!", 需要在显示字符串后延迟一段时间, 然后再清除掉显示的字符串。显示字符串可以使用功能号 ah=13h 的 int 10h 功能调用。延迟函数 Delay 如下:

```

Delay:                                ; 延迟一段时间
    push ax
    push cx
    mov ax, 580
delay_outer:
    mov cx, 50000
delay_inner:
    loop delay_inner
    dec ax
    cmp ax, 0
    jne delay_outer
    pop cx
    pop ax
    ret

```

然后改动实验3中的那四个用户程序: 在开始处转移中断向量, 并写入中断向量表:

```
MOVE_INT_VECTOR 09h, 39h
WRITE_INT_VECTOR 09h, IntTouch
```

在用户程序退出前恢复中断向量，不影响内核以及其他程序使用 BIOS 自己的 int 09h：

```
QuitUsrProg:
    MOVE_INT_VECTOR 39h, 09h
```

(3) `macro.asm`：该文件将写中断向量表、转移中断向量、在固定位置打印字符这三个操作实现为宏：

```
%macro WRITE_INT_VECTOR 2          ;写中断向量表
%macro MOVE_INT_VECTOR 2           ;将参数1的中断向量转移至参数2处
%macro PRINT_IN_POS 4
```

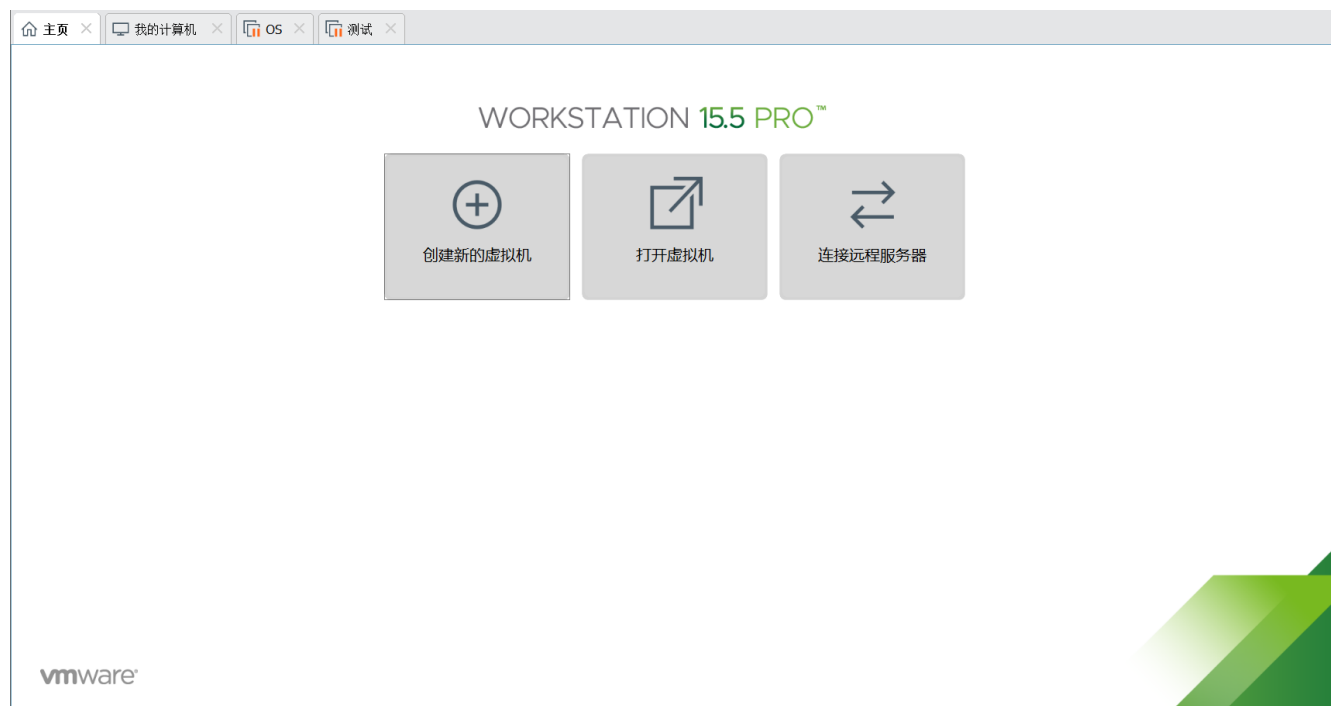
四、实验过程：

1.软件的下载与安装：

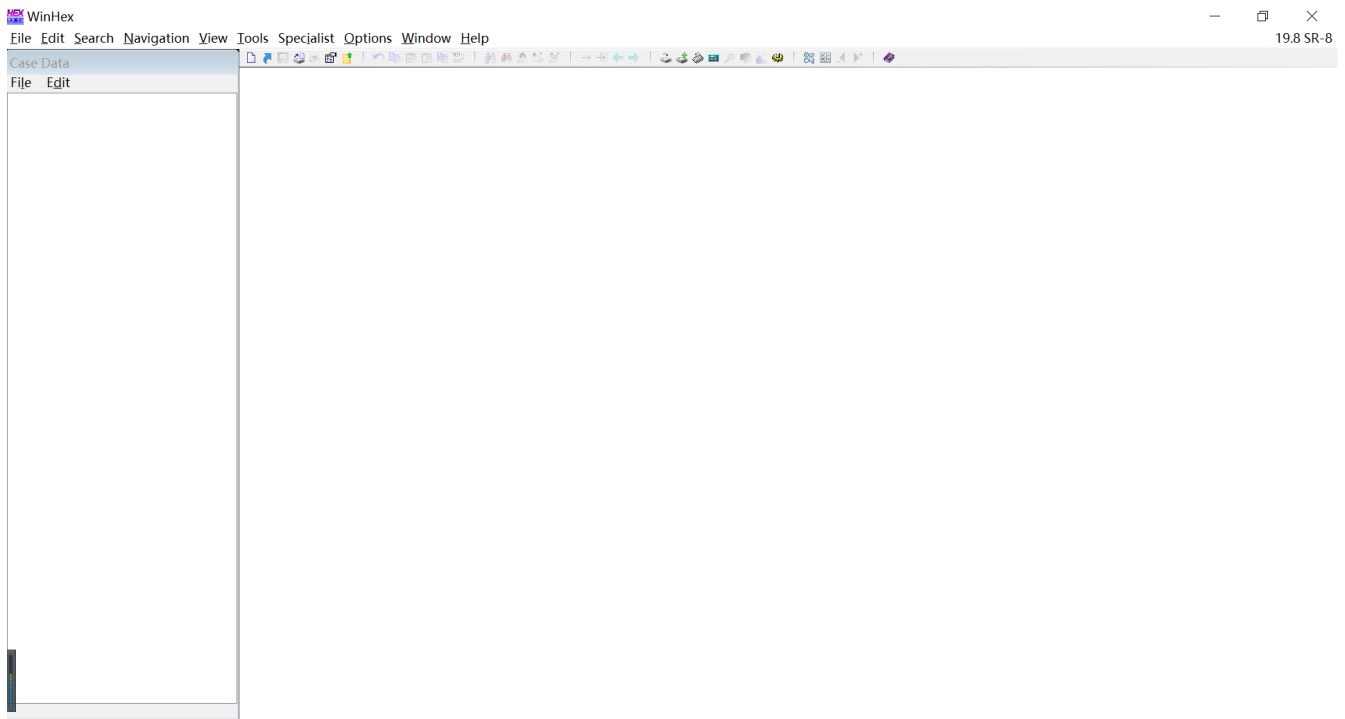
(1) 在官网下载 VMware Workstation 15.5 Pro：

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

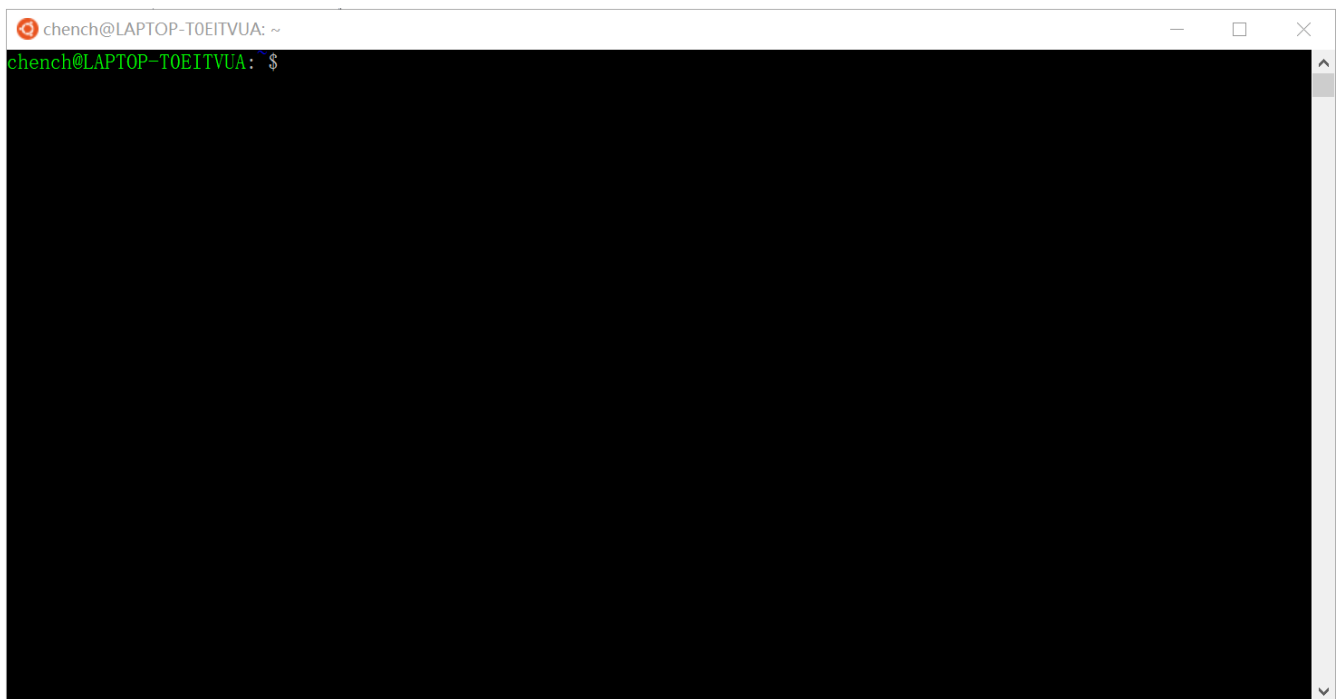
安装该软件后初始界面如下（已经输入密钥并初步设置后）：



(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，输入指令 `explorer.exe` . 进入 `WSL` 的目录，将编写好的代码放在 `WSL` 目录下：

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

| 名称 | 修改日期 | 类型 | 大小 |
|---|-----------------|-----------------|-------|
|  .bash_history | 2020/5/22 21:45 | BASH_HISTORY 文件 | 17 KB |
|  intouch.asm | 2020/5/22 21:44 | ASM 文件 | 2 KB |
|  ka.asm | 2020/5/22 21:44 | ASM 文件 | 10 KB |
|  kc.c | 2020/5/22 21:44 | C Source File | 7 KB |
|  kernel.asm | 2020/5/22 21:44 | ASM 文件 | 1 KB |
|  macro.asm | 2020/5/22 21:44 | ASM 文件 | 2 KB |
|  stonelu.asm | 2020/5/22 21:44 | ASM 文件 | 5 KB |
|  stringio.h | 2020/5/22 21:44 | C Header File | 3 KB |
|  usrproginfo.asm | 2020/5/22 21:44 | ASM 文件 | 2 KB |
|  bootloader.asm | 2020/5/22 21:44 | ASM 文件 | 3 KB |
|  hotwheel.asm | 2020/5/22 21:44 | ASM 文件 | 2 KB |
|  stonerd.asm | 2020/5/22 21:44 | ASM 文件 | 5 KB |
|  stoneru.asm | 2020/5/22 21:44 | ASM 文件 | 5 KB |
|  stoneld.asm | 2020/5/22 21:44 | ASM 文件 | 5 KB |

类型: ASM 文件
大小: 504 字节
修改日期: 2020/5/22 21:44

其次，进行编译链接：（如何编译链接与实验三中相同，此处就不再赘述）

```
chench@LAPTOP-T0EITVUA: $ nasm -f bin usrproginfo.asm -o usrproginfo.bin
chench@LAPTOP-T0EITVUA: $ nasm -f bin bootloader.asm -o bootloader.bin
chench@LAPTOP-T0EITVUA: $ nasm -f bin stoneld.asm -o stoneld.bin
chench@LAPTOP-T0EITVUA: $ nasm -f bin stonerd.asm -o stonerd.bin
chench@LAPTOP-T0EITVUA: $ nasm -f bin stoneru.asm -o stoneru.bin
chench@LAPTOP-T0EITVUA: $ nasm -f bin stonelu.asm -o stonelu.bin
chench@LAPTOP-T0EITVUA: $ nasm -f elf32 hotwheel.asm -o hotwheel.o
chench@LAPTOP-T0EITVUA: $ nasm -f elf32 kernel.asm -o kernel.o
chench@LAPTOP-T0EITVUA: $ nasm -f elf32 ka.asm -o ka.o
chench@LAPTOP-T0EITVUA: $ gcc -march=i386 -m16 -mpreferred-stack-boundary=2 -ffreestanding -fno-PIE -masm=intel -c kc.c -o kc.o
chench@LAPTOP-T0EITVUA: $ ld -m elf_i386 -N --oformat binary -Ttext 0x8000 kernel.o ka.o kc.o hotwheel.o -o kernel.bin
chench@LAPTOP-T0EITVUA: $
```

编译完成后可以在该目录下找到上述所有 .bin 文件。

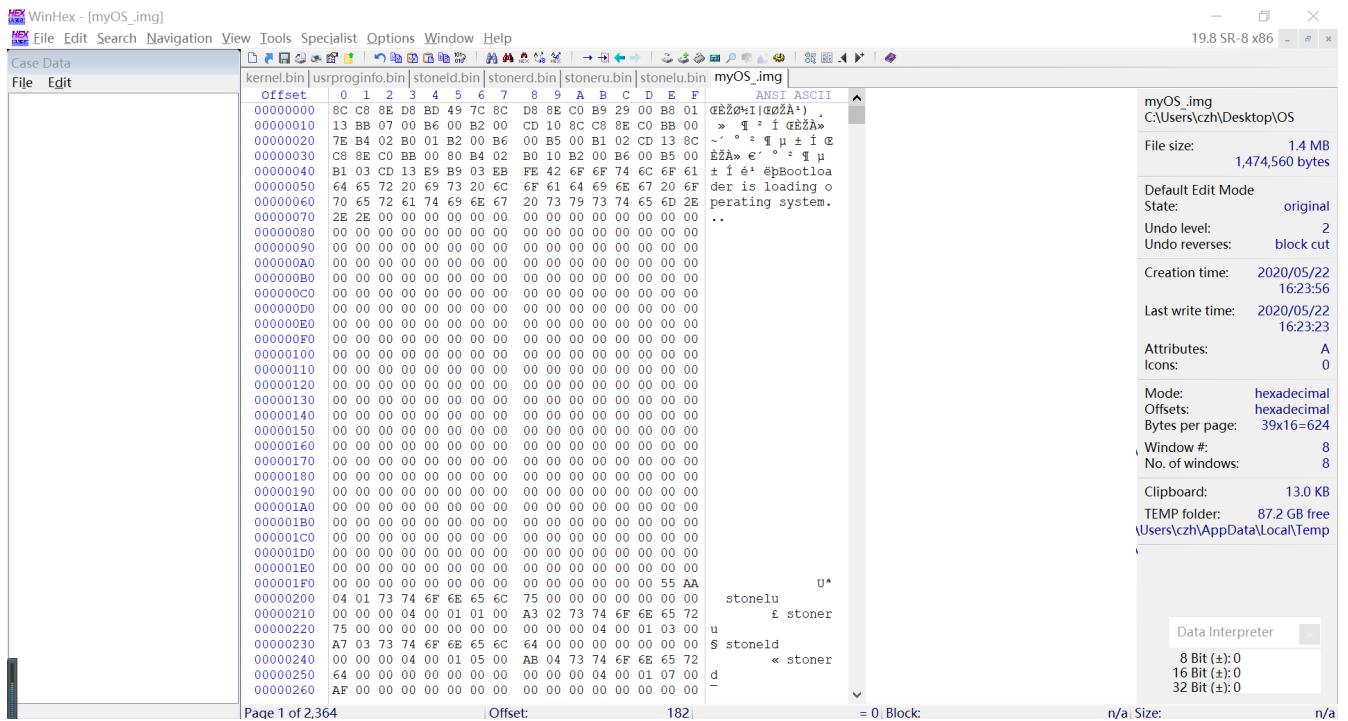
最后，便是得到本次实验所需的镜像文件：

将实验3中生成的镜像文件 myOS.img 复制一份并在 winHex 内打开，用上一步新生成的 .bin 文件将原文件覆盖后即可得到本次实验的镜像文件，命名为 myOS_.img。

打开制作软盘所需的所有文件（共7个），然后将这7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->stoneld.bin->stonerd.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，oskernel.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)

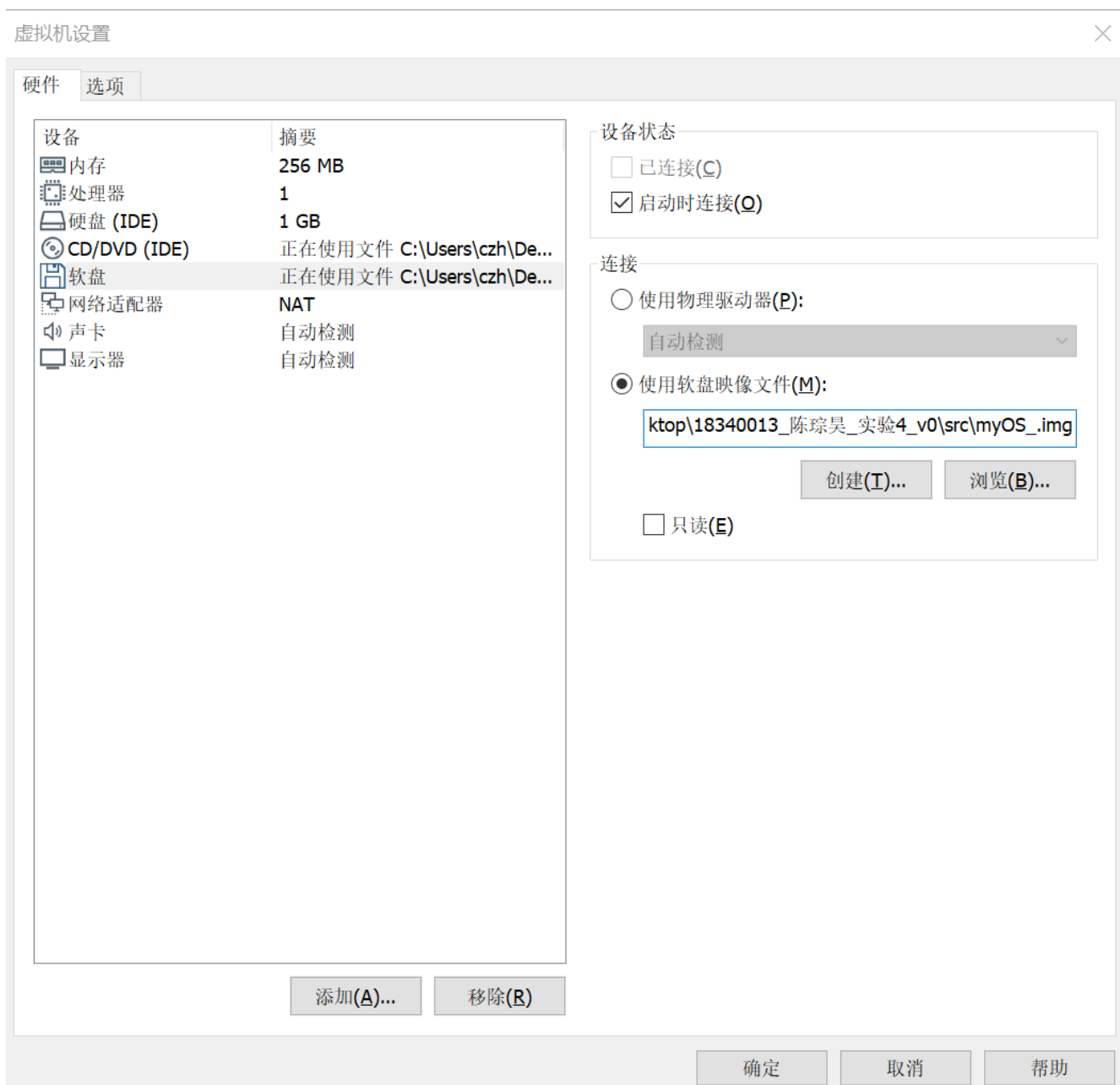
下一步(N) >

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：

```
This is myOS. These commands can be executed. Use 'help' to see the list.  
  help - show information about myOS  
  clear - clear the screen  
  list - show a list of user program and their PIDs  
  run <PIDs> - run user program in sequence, e.g. 'run 4 3 2 1'  
  poweroff - shutdown the machine  
  date - display the current date and time  
  onoff - turn on/off the hotwheel  
myOS>>_
```

可以看到显示了彩色的提示信息。

(3) 执行 `help` 操作：

```
This is myOS. These commands can be executed. Use 'help' to see the list.  
  help - show information about myOS  
  clear - clear the screen  
  list - show a list of user program and their PIDs  
  run <PIDs> - run user program in sequence, e.g. 'run 4 3 2 1'  
  poweroff - shutdown the machine  
  date - display the current date and time  
  onoff - turn on/off the hotwheel
```

```
myOS>>help
```

```
This is myOS. These commands can be executed. Use 'help' to see the list.  
  help - show information about myOS  
  clear - clear the screen  
  list - show a list of user program and their PIDs  
  run <PIDs> - run user program in sequence, e.g. 'run 4 3 2 1'  
  poweroff - shutdown the machine  
  date - display the current date and time  
  onoff - turn on/off the hotwheel
```

```
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:

```
myOS>>list
You can use 'run <PID>' to run a user program.
PID - Name - Size - Addr - Cylinder - Head - Sector
1 - stonelu - 1024 - A300 - 0 - 1 - 1
2 - stoneru - 1024 - A700 - 0 - 1 - 3
3 - stoneld - 1024 - AB00 - 0 - 1 - 5
4 - stonerd - 1024 - AF00 - 0 - 1 - 7
myOS>>
```

显示了4个用户程序的信息。

(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。在执行多个程序过程中, 中途可按 e 退出进入下一程序。在按键的时候会执行中断, 显示 OUCH!OUCH!。所有程序执行完毕后会返回到该界面:

```
All program have been executed successfully in sequence.
myOS>>_
```

注: 该部分结合录屏观看比较好, 录屏在 src 文件夹的录屏展示里。

(7) 执行 onoff 操作: 默认情况下会在右下角显示"风火轮", 执行一次 onoff 则关闭"风火轮", 再执行一次 onoff 则又重新开启。该操作也在录屏中有展示, 这里简单截取两个画面:

```
All program have been executed successfully in sequence.  
myOS>>onoff  
Hotwheel has been turned off.  
myOS>>
```

执行一次 onoff 可以看到"风火轮"没有了。

```
All program have been executed successfully in sequence.  
myOS>>onoff  
Hotwheel has been turned off.  
myOS>>onoff  
Hotwheel has been turned on.  
myOS>>_
```

再执行一次 onoff 可以看到"风火轮"又出现了。

(8) 执行 date 指令：显示当前时间。


```
myOS>>help
This is myOS. These commands can be executed. Use 'help' to see the list.
help - show information about myOS
clear - clear the screen
list - show a list of user program and their PIDs
run <PIDs> - run user program in sequence, e.g. 'run 4 3 2 1'
poweroff - shutdown the machine
date - display the current date and time
onoff - turn on/off the hotwheel
myOS>>date
2020/5/22 22:22:19
myOS>>_
```

(9) 执行 `poweroff` 操作：直接关闭虚拟机。



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

```
This is myOS. These commands can be executed. Use 'help' to see the list.
help - show information about myOS
clear - clear the screen
list - show a list of user program and their PIDs
run <PIDs> - run user program in sequence, e.g. 'run 4 3 2 1'
poweroff - shutdown the machine
date - display the current date and time
onoff - turn on/off the hotwheel
myOS>>aaaaa
aaaaa: command not found
myOS>>
```

五、实验总结：

本次实验相对来讲比上次实验要轻松。因为本次实验就是在上次实验的基础上添加功能，使其更加完善；肯定是要比从头开始实现内核要简单一些。说其"轻松"的很重要的一个原因就在于需要解决的问题比上次实验少。这次实验只需要添加中断就可以了；而上次实验涉及C与汇编的互相调用，以及编译、链接等等。在这次实验里这两个问题已经不算问题了，都是相同的套路。关键还是在于汇编代码的编写，C语言模块只是添加进去一些接口就可以，并能够实现与汇编的互相调用；但调用相关的功能还是要通过汇编去实现，如果汇编没有掌握好的话那么完成这次任务也不简单。本次实验的关键就是理解时间中断与键盘中断。为了理解清楚这两个概念以及其实现的原理，需要学习很多东西，我在网上也查找了不少资料，也咨询了同学，可以说本次实验把很长时间都花在了学习这些知识上面，弄清楚理论以后写代码还真没花多长时间，后面编译链接等自然是水到渠成。不过在测试的时候，我不经意间发现了一个小问题：键盘中断时好像会显示2次"OUCH! OUCH!"，即按键的时候会显示，松开的时候也会显示（尽管只是一瞬间的事但还是被我发现了）。出现这种效果的原因是当按下按键或松开按键都会触发 `int 09h` 中断，产生的扫描码后者比前者大 `80h`，这也就是为什么在用户程序执行时每次按下键盘和松开键盘都会显示"OUCH! OUCH!"的原因。当按下后立即松开，即是在短时间内两次触发了 `int 09h` 中断，因此会显示2次。

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. 03实验课.pptx

https://blog.csdn.net/qq_23880193/article/details/43274599

https://blog.csdn.net/qq_37232329/article/details/79895350?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.nonecase

<https://wenku.baidu.com/view/ae862524b9f3f90f77c61b72.html>

