

## 武汉大学国家网络安全学院教学实验报告

课程名称	操作系统设计与实践	实验日期	2023. 10. 09
实验名称	实验环境搭建：搭建实验的基本环境，熟悉开发与调试工具	实验周次	第一周
姓名	学号	专业	班级
王卓	2021302191791	网络空间安全	9 班
程子洋	2021301051114	网络空间安全	9 班
聂森	2021302191536	网络空间安全	9 班
刘琥	2021302121234	网络空间安全	9 班
<b>一、实验目的及实验内容</b> (本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析)			
<b>实验知识要求：</b> 1. 操作系统启动过程： 需要理解计算机启动时的基本流程，包括 BIOS 的初始化、硬件自检、引导加载程序的执行以及操作系统内核的启动； 2. 虚拟化技术：需要了解虚拟机技术，包括如何使用虚拟化软件（如 VirtualBox，VMware）创建虚拟机、安装操作系统，以及在虚拟环境中进行开发和调试； 3. 操作系统开发环境： 需要学习如何在 Ubuntu 操作系统中安装必要的开发环境，包括编译工具链和调试工具； 4. Bochs 模拟器： 需要掌握 Bochs 模拟器的使用，包括如何下载、编译和配置 Bochs，以及如何运行虚拟机并进行调试； Bochs 调试的基本命令 ·设置断点 b address ·显示所有断点 info break ·继续执行 c ·单步执行 s （可跳入函数） n（跳过函数内部） ·反汇编命令 u 起始地址 终止地址 ·查看通用寄存器信息 r，段寄存器 sreg，控制寄存器 creg 5. 汇编语言编程： 需要了解 x86 汇编语言的基本语法和概念； 6. 反汇编： 需要学会使用反汇编工具查看汇编代码，并理解汇编代码与机器代码之间的关系； 7. 调试技能： 需要掌握 Bochs 调试器的基本命令，包括设置断点、单步执行、查看寄存器状态等，以便调试自己编写的引导程序； 8. 段寄存器的作用： 需要理解为什么需要对段寄存器进行赋值，以确保程序正确访问内存； 9. 重复执行： 需要思考如何修改引导程序，以便让输出过程执行多次，例如使用循环结构。  <b>实验内容：</b> 1. 认真阅读章节材料 2. 在实验机上安装虚拟运行环境，并安装 ubuntu			

3. 安装 ubuntu 开发环境，32 位环境
4. 下载 bochs 源码，编译并安装 bochs 环境
5. 使用 bochs 自带工具 bximage 创建虚拟软驱
6. 阅读、编译 boot.asm，并反汇编阅读
7. 修改 bochsrc，运行并调试你的第一个程序
8. 完成实验练习要求
  - (1) 删除 0xAA55，观察程序效果，找出原因
  - (2) 修改程序中输出为，一个包含自己名字的字符串，调试程序
  - (3) 把生成的可执行文件反汇编，看看输出的内容是怎样的，并在虚拟机启动过程，设置断点进行调试，在实验报告中截图
  - (4) 为什么要 jmp \$，如何改造程序，让这个输出过程执行 100 次
  - (5) 回答：为什么要对段寄存器进行赋值
  - (6) 回答：如何在程序中调用系统中断

#### 原理分析：

1. 操作系统启动过程：了解计算机加电后的 BIOS 启动流程，包括硬件初始化和引导加载程序的执行；
2. 虚拟机环境：学习如何使用虚拟机软件（如 VirtualBox，VMware），创建虚拟机，安装操作系统，并配置开发环境；
3. Bochs 调试器：了解如何使用 Bochs 调试器进行操作系统开发的调试工作，包括设置断点、查看寄存器状态等；
4. 汇编语言编程：学习如何编写汇编语言程序，理解汇编代码与机器代码的关系；
5. 引导程序：理解引导程序，将其加载到虚拟软驱中，并设置 Bochs 以执行引导程序。

## 二、实验环境及实验步骤

（本次实验所使用的器件、仪器设备等的情况；具体的实验步骤）

### 实验环境：

- 虚拟化软件：VMware Workstation Pro/VMware Workstation 16 player
- Ubuntu 16.04
- Bochs 2.7

### 实验步骤：

#### 一、阅读章节资料

在开始实验之前，仔细阅读提供的章节资料，复习操作系统启动过程的基本原理和实验的大致内容。

#### 二、通过[官网](#)安装虚拟机 VMware，并安装 Ubuntu16.04 32 位环境

1. 拷贝 Ubuntu16.04 镜像，在 VMware Workstation Pro/VMware Workstation 16player 中安装 32 位版本的 Ubuntu 操作系统；
2. 合理分配虚拟机资源；
3. 根据实验要求进行软件源修改，以加快软件包的下载和安装速度。

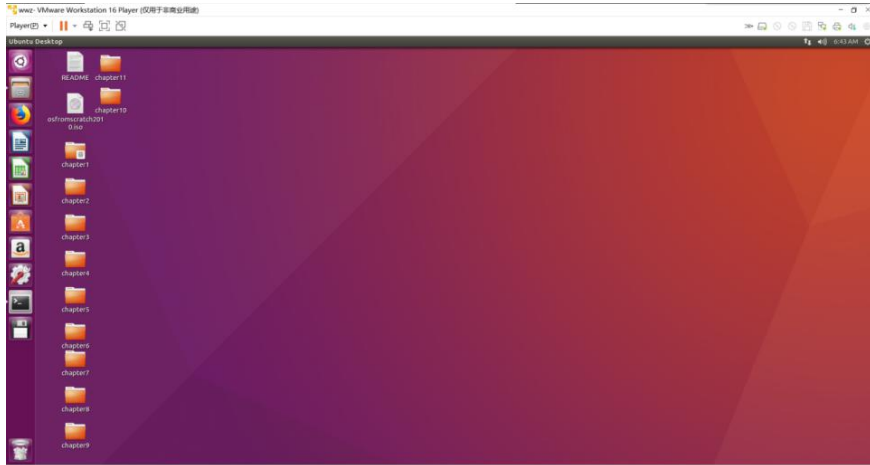


图 1 虚拟机成功安装展示

注：篇幅原因，仅展示一张图片以示例

#### 4. 安装开发环境

在 Ubuntu 操作系统中安装必要的开发环境，包括编译工具链（如 gcc）、nasm 汇编编译器，使用以下命令来安装这些依赖项：

1. `sudo apt-get update`
2. `sudo apt-get install gcc`
3. `sudo apt-get install nasm`

### 三、下载 bochs 源码，编译并安装 bochs 环境

#### 1. 官网下载 bochs 源码

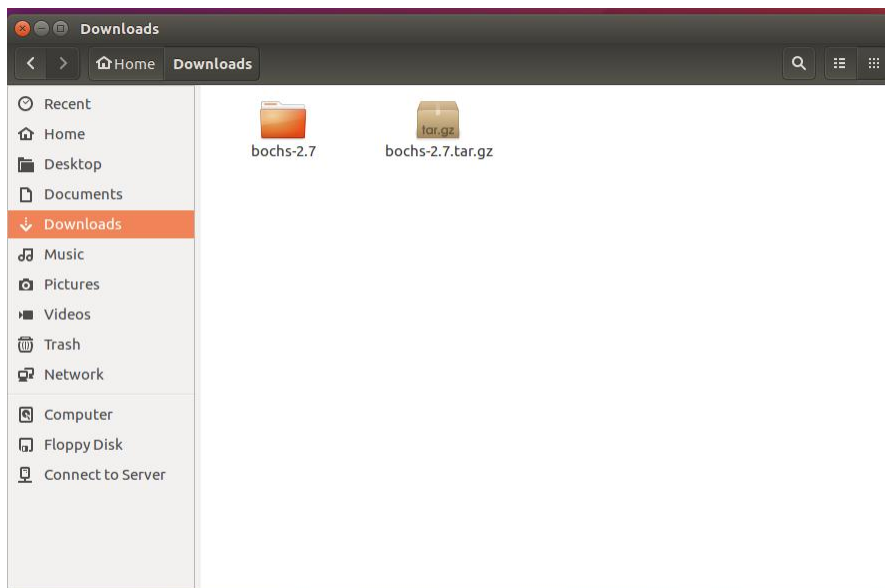


图 2 bochs 源码成功下载展示

注：篇幅原因，仅展示一张图片以示例

#### 2. 解压源码包并进入解压后的目录



```
shizi@shizi-VirtualBox:~/下载/bochs-2.7$ make
cd iodev && \
make libiodev.a
make[1]: Entering directory '/home/shizi/下载/bochs-2.7/iodev'
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g
_OFFSET_BITS=64 -D_LARGE_FILES -pthread devices.cc -o devices.o
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g
_OFFSET_BITS=64 -D_LARGE_FILES -pthread virt_timer.cc -o virt_tim
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g
_OFFSET_BITS=64 -D_LARGE_FILES -pthread slowdown_timer.cc -o slow
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g
_OFFSET_BITS=64 -D_LARGE_FILES -pthread pic.cc -o pic.o
g++ -c -I.. -I../.. -I../instrument/stubs -I../instrument/stubs -g
_OFFSET_BITS=64 -D_LARGE_FILES -pthread pit.cc -o pit.o
```

图 5 安装展示

注：篇幅原因，仅展示一张图片以示例

#### 四、使用 bochs 自带工具 bximage 创建虚拟软驱

1. `sudo bximage`

```
wz@ubuntu:~/Downloads$ bximage
=====
bximage
Disk Image Creation / Conversion / Resize and Commit Tool for Bochs
$Id: bximage.cc 14091 2021-01-30 17:37:42Z sshwartz $
=====
1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info
0. Quit
Please choose one [0] 1
Create image
Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] fd
Choose the size of floppy disk image to create.
Please type 160k, 180k, 320k, 360k, 720k, 1.2M, 1.44M, 1.68M, 1.72M, or 2.88M.
[1.44M]
What should be the name of the image?
[a.img]
Creating floppy image 'a.img' with 2880 sectors
The following line should appear in your bochsrc:
floppya: image="a.img", status=inserted
```

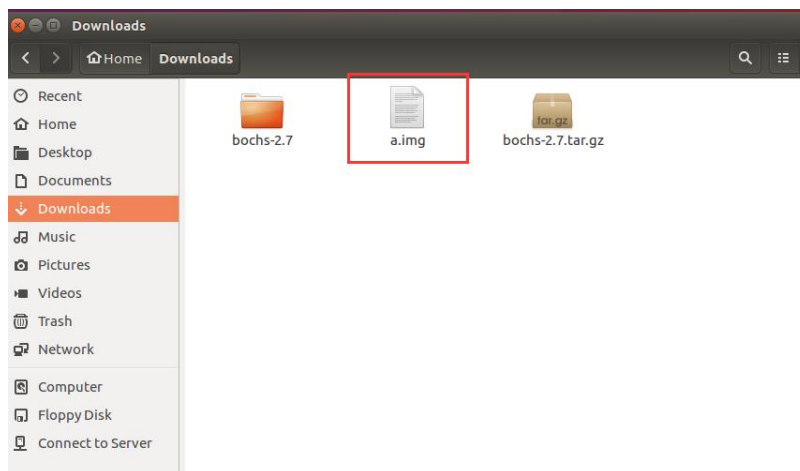


图 6 创建虚拟软驱

注：篇幅原因，仅展示两张图片以示例

生成虚拟软驱 a.img 如上图所示。

## 五、阅读、编译 boot.asm，并反汇编阅读

### 1. 阅读 boot.asm

```
Open ▾ 
org 07c00h ; 告诉编译器程序加载到7c00处
mov ax, cs
mov ds, ax
mov es, ax
call DispStr ; 调用显示字符串例程
jmp $ ; 无限循环
DispStr:
mov ax, BootMessage
mov bp, ax ; ES:BP = 串地址
mov cx, 16 ; CX = 串长度
mov ax, 01301h ; AH = 13, AL = 01h
mov bx, 000ch ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)
mov dl, 0
mov int 10h ; 10h 号中断
ret
BootMessage: db "Hello, louis!"
times 510-($-$$) db 0 ; 填充剩下的空间，使生成的二进制代码恰好为512字节
dw 0xaa55 ; 结束标志
```

图 7 boot.asm 文件内容展示

注：篇幅原因，仅展示一张图片以示例

### 2. 编译 boot.asm

使用 nasm 编译 boot.asm 文件，并生成引导文件 boot.bin，命令如下：

1. nasm boot.asm -o boot.bin #生成引导文件

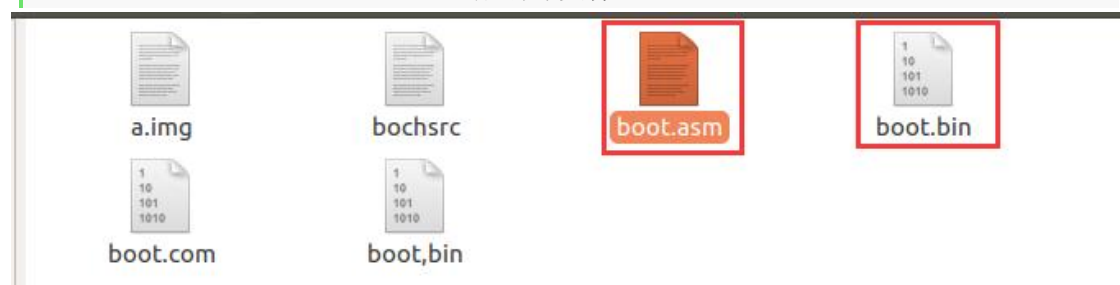


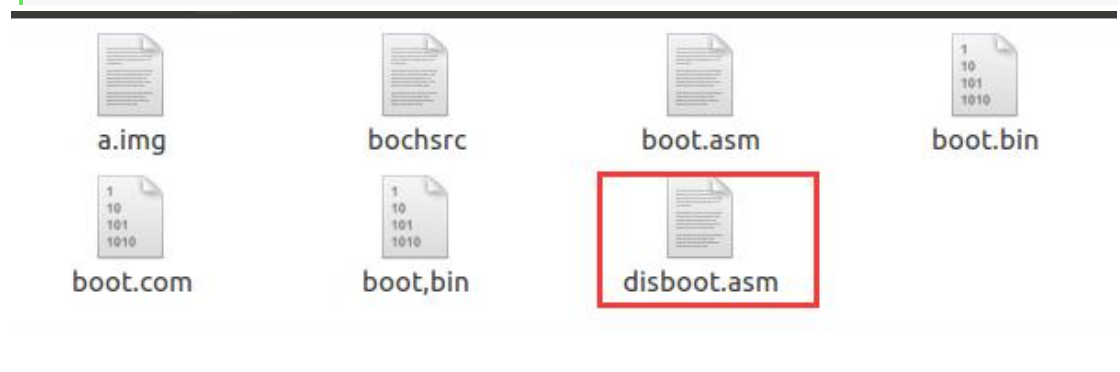
图 8 编译 boot.asm 文件

注：篇幅原因，仅展示一张图片以示例

### 3. 反汇编阅读

使用 ndisasm 反汇编引导文件 boot.bin，生成 disboot.asm，命令如下：

1. sudo ndisasm -o 0x7c00 boot.bin >> disboot.asm





Open	Address	Disassembly
	00007C00	8CC8 mov ax,cs
	00007C02	8ED8 mov ds,ax
	00007C04	8EC0 mov es,ax
	00007C06	E80200 call word 0x7c0b
	00007C09	EBFE jmp short 0x7c09
	00007C0B	B81E7C mov ax,0x7c1e
	00007C0E	89C5 mov bp,ax
	00007C10	B91000 mov cx,0x10
	00007C13	B80113 mov ax,0x1301
	00007C16	B80C00 mov bx,0xc
	00007C19	B200 mov dl,0x0
	00007C1B	CD10 int 0x10
	00007C1D	C3 ret
	00007C1E	48 dec ax
	00007C1F	656C gs insb
	00007C21	6C insb
	00007C22	6F outsw
	00007C23	2C20 sub al,0x20
	00007C25	4F dec di
	00007C26	53 push bx
	00007C27	20776F and [bx+0x6f],dh
	00007C2A	726C jc 0x7c98
	00007C2C	642100 and [fs:bx+si],ax
	00007C2F	0000 add [bx+si],al
	00007C31	0000 add [bx+si],al
	00007C33	0000 add [bx+si],al
	00007C35	0000 add [bx+si],al
	00007C37	0000 add [bx+si],al
	00007C39	0000 add [bx+si],al
	00007C3B	0000 add [bx+si],al
	00007C3D	0000 add [bx+si],al
	00007C3F	0000 add [bx+si],al
	00007C41	0000 add [bx+si],al
	00007C43	0000 add [bx+si],al
	00007C45	0000 add [bx+si],al
	00007C47	0000 add [bx+si],al
	00007C49	0000 add [bx+si],al
	00007C4B	0000 add [bx+si],al
	00007C4D	0000 add [bx+si],al
	00007C4F	0000 add [bx+si],al
	00007C51	0000 add [bx+si],al
	00007C53	0000 add [bx+si],al
	00007C55	0000 add [bx+si],al
	00007C57	0000 add [bx+si],al
	00007C59	0000 add [bx+si],al
	00007C5B	0000 add [bx+si],al
	00007C5D	0000 add [bx+si],al
	00007C5F	0000 add [bx+si],al
	00007C61	0000 add [bx+si],al
	00007C63	0000 add [bx+si],al

图 9 反汇编 boot.bin 文件

注：篇幅原因，仅展示两张图片以示例

## 六、修改 bochsrc，运行并调试第一个汇编程序

### 1. 修改 bochsrc

修改 vgaromimage 和 romimage 为对应的 bochs 安装后的位置；注释掉 keyboard\_mapping 一行；

```
# filename of ROM images
romimage: file=/home/wz/Downloads/bochs-2.7/bios/BIOS-bochs-latest
vgaromimage: file=/home/wz/Downloads/bochs-2.7/bios/VGABIOS-lgpl-latest
```

图 10(a) 修改路径(王卓)

```
# filename of ROM images
romimage: file=/usr/local/share/bochs/BIOS-bochs-latest
vgaromimage: file=/usr/local/share/bochs/VGABIOS-lgpl-latest
```

图 10(b) 修改路径(程子洋)

```
# filename of ROM images
romimage: file=/home/shizi/Desktop/bochs-2.7/bios/BIOS-bochs-latest
vgaromimage: file=/usr/share/vgabios/vgabios.bin
```

图 10(c) 修改路径(刘虢)

图 10 修改路径

该文件设定了启动的软盘为 a.img

## 2. 运行程序

### (1) 写引导盘

在前面已完成源码的编译，虚拟软驱的创建之后，需要写引导盘，将 boot.bin 中的数据复制到 a.img 文件的首个扇区中，而不影响 a.img 文件的其他部分。命令如下：

1. `sudo dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc`

```
00452044000i[ SIM ] quit SIM called with exit code 1
wz@ubuntu:~/Desktop/chapter1/a$ sudo dd if=boot.bin of=a.img bs=512 count=1 conv
=notrunc
0+1 records in
0+1 records out
510 bytes copied, 0.000108095 s, 4.7 MB/s
wz@ubuntu:~/Desktop/chapter1/a$ bochs
```

图 11(a) 写引导盘(王卓)

```
muye@muye-virtual-machine:~/OS/osfs01-master$ nasm boot.asm -o boot.bin
muye@muye-virtual-machine:~/OS/osfs01-master$ dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
记录了0+1 的读入
记录了0+1 的写出
510 bytes copied, 0.00102311 s, 498 kB/s
muye@muye-virtual-machine:~/OS/osfs01-master$
```

图 11(b) 写引导盘(程子洋)

```
shizi@shizi-virtual-machine:~/Desktop/chapter1/a$ dd if=boot.bin of=a.img bs=512
count=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000103086 s, 5.0 MB/s
```

图 11(c) 写引导盘(刘琥)

图 11 写引导盘

### (2) 运行

1. `sudo bochs -f ./bochsrc`

```
wz@ubuntu: ~/Desktop/chapter1/a
wz@ubuntu:~/Desktop/chapter1/a$ sudo bochs -f bochsrc
[sudo] password for wz:
=====
Bochs x86 Emulator 2.7
Built from SVN snapshot on August 1, 2021
Timestamp: Sun Aug 1 10:07:00 CEST 2021
=====
00000000000i[      ] BXSHARE not set. using compile time default '/usr/local/sha
re/bochs'
00000000000i[      ] reading configuration from bochsrc
-----
Bochs Configuration: Main Menu
-----

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate. Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found. When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

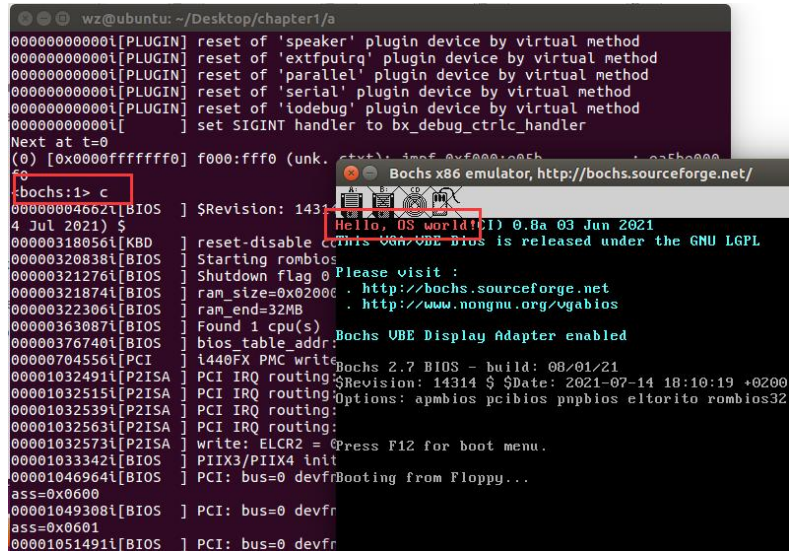
Please choose one: [6]
```

图 12 运行引导程序 boot.asm

注：篇幅原因，仅展示一张图片以示例

接着选择[6]开始 simulation，输入 c 表示继续，得到 Hello, OS world，成功。





```
wz@ubuntu: ~/Desktop/chapter1/a
00000000000i[PLUGIN] reset of 'speaker' plugin device by virtual method
00000000000i[PLUGIN] reset of 'extfpurq' plugin device by virtual method
00000000000i[PLUGIN] reset of 'parallel' plugin device by virtual method
00000000000i[PLUGIN] reset of 'serial' plugin device by virtual method
00000000000i[PLUGIN] reset of 'idebug' plugin device by virtual method
00000000000i[ ] set SIGINT handler to bx_debug_ctrlc_handler
Next at t=0
(0) [0x0000fffff0] f000:fff0 (unk.
f0
bochs:1> c
00000004662i[BIOS] $Revision: 14314 $ $Date: 2021-07-14 18:10:19 +0200
4 Jul 2021) $
00000318056i[KBD] ] reset-disable (This VGA VBE Bios is released under the GNU LGPL
00000320838i[BIOS] ] Starting rombios
00000321276i[BIOS] ] Shutdown flag 0
00000321874i[BIOS] ] ram_size=0x02000
00000322306i[BIOS] ] ram_end=32MB
00000363087i[BIOS] ] Found 1 cpu(s)
00000376740i[BIOS] ] bios_table_addr:
00000704556i[PCI] ] i440FX PMC write
00001032491i[P2ISA] ] PCI IRQ routing
00001032515i[P2ISA] ] PCI IRQ routing
00001032539i[P2ISA] ] PCI IRQ routing
00001032563i[P2ISA] ] PCI IRQ routing
00001032573i[P2ISA] ] write: ELCR2 = 0Press F12 for boot menu.
00001033342i[BIOS] ] PIIX3/PIIX4 init
00001046964i[BIOS] ] PCI: bus=0 devfrBooting from Floppy...
ass=0x0600
00001049308i[BIOS] ] PCI: bus=0 devfr
ass=0x0601
00001051491i[BIOS] ] PCI: bus=0 devfr
Hello, OS world!
Please visit :
http://bochs.sourceforge.net
http://www.nongnu.org/vgabios
Bochs VBE Display Adapter enabled
Bochs 2.7 BIOS - build: 08/01/21
$Revision: 14314 $ $Date: 2021-07-14 18:10:19 +0200
Options: apmbios pcibios pnpbios eltorito rombios32
```

图 13 运行结果展示

注：篇幅原因，仅展示一张图片以示例

## 七、实验练习

### 1. 删除 0xAA55，观察程序效果，找出原因；

(1) 删除 0xAA55 后的 boot.asm 文件



```
boot.asm (~/.OS) - gedit
Open Save

org 07c00h ; 告诉编译器程序加载到7c00处
mov ax, cs
mov ds, ax
mov es, ax
call DispStr ; 调用显示字符串例程
jmp $ ; 无限循环
DispStr:
mov ax, BootMessage
mov bp, ax ; ES:BP = 串地址
mov cx, 16 ; CX = 串长度
mov ax, 01301h ; AH = 13, AL = 01h
mov bx, 000ch ; 页号为0(BH = 0) 黑底红字(BL = 0ch,高亮)
mov dl, 0
int 10h ; 10h 号中断
ret
BootMessage: db "Hello, OS world!"
times 510-($-$$) db 0 ; 填充剩下的空间,使生成的二进制代码恰好为512字节
```

图 14 修改后 boot.asm 文件展示

注：篇幅原因，仅展示一张图片以示例

(2) 根据前文操作编译运行程序

1. sudo nasm boot.asm -o boot.bin
2. sudo bxiimage
3. sudo dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
4. sudo bochs -f ./bochsrc

图 15(a) 练习 1(王卓)

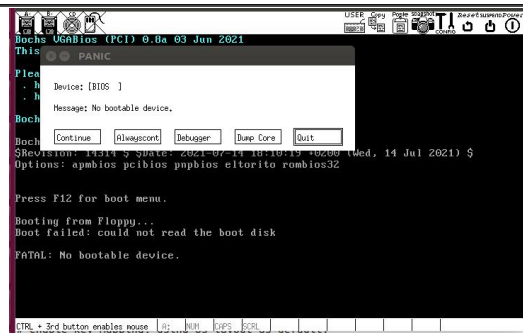


图 15(b) 练习 1(程子洋)

图 15(c) 练习 1(聂森)

图 15(d) 练习 1(刘斌)

图 15 练习 1 error

发现出现错误为“No bootable device”

原因分析:

当计算机电源打开时, 会首先进行加电自检(POST), 然后寻找启动盘, 这里我们设置的是从软盘启动, 所以计算机会检查软盘的 0 面 0 磁道 1 扇区, 当其发现以 0xAA55 结束时, BIOS 就会认定其为引导扇区, 然后将 1 扇区的 512 字节的内容装载到内存地址 0000: 7c00 处。当删除了 0xAA55 后, 即删除了引导标志, 会导致引导扇区不再被认为是有效的引导扇区, 因此计算机将无法正确启动, 后续操作无法进行, 通常会显示一个错误信息或者停留在一个错误的状态下, 与报错信息“无引导设备”含义一致。

## 2. 修改程序中输出为, 一个包含自己名字的字符串, 调试程序;

观察 boot.asm 文件可以发现输出的字符串即 BootMessage“ ”中的内容, 故将“Hello, OS world”修改为自己名字即可。

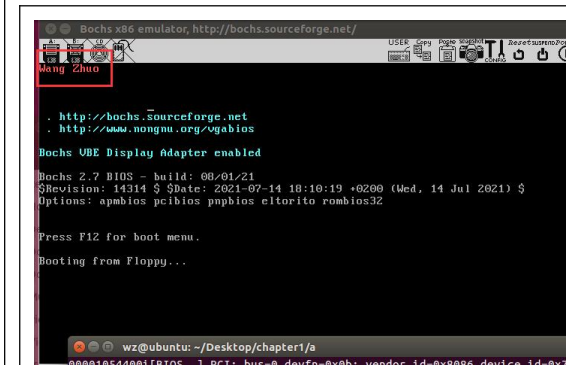


图 16(a) 练习 2(王卓)

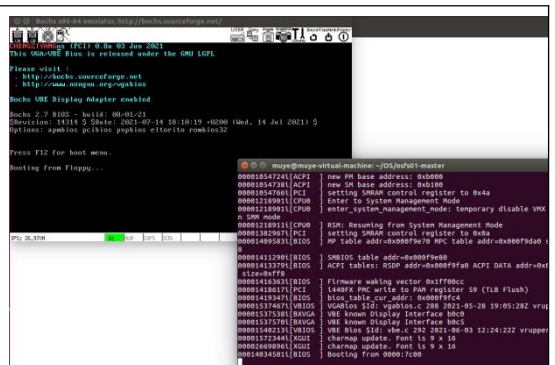


图 16(b) 练习 2(程子洋)



图 16(c) 练习 2(聂森)

图 16(d) 练习 2(刘虢)

图 16 练习 2 代码运行结果

注：在王卓的实验中，cx 的值修改为寄存器的极限值 512，故出现大量空白

3. 把生成的可执行文件反汇编，看看输出的内容是怎样的，并在虚拟机启动过程，设置断点进行调试；

### ① 王卓：

(1) 可执行文件反汇编

执行命令：sudo ndisasm -o 0x7c00 boot.bin >> disboot.asm

得到输出内容为：

Open	IP	
00007C00	8CC8	mov ax,cs
00007C02	8ED8	mov ds,ax
00007C04	8EC0	mov es,ax
00007C06	E80200	call word 0x7c0b
00007C09	E8FE	jmp short 0x7c09
00007C0B	B81E7C	mov ax,0x7c1e
00007C0E	89C5	mov bp,ax
00007C10	B91000	mov cx,0x10
00007C13	B80113	mov ax,0x1301
00007C16	B80C00	mov bx,0xc
00007C19	B200	mov dl,0x0
00007C1B	CD10	int 0x10
00007C1D	C3	ret
00007C1E	48	dec ax
00007C1F	656C	gs insb
00007C21	6C	insb
00007C22	6F	outsw
00007C23	2C20	sub al,0x20
00007C25	4F	dec di
00007C26	53	push bx
00007C27	20776F	and [bx+0x6f],dh
00007C2A	726C	jc 0x7c98
00007C2C	642100	and [fs:bx+si],ax
00007C2F	0000	add [bx+si],al
00007C31	0000	add [bx+si],al
00007C33	0000	add [bx+si],al
00007C35	0000	add [bx+si],al
00007C37	0000	add [bx+si],al
00007C39	0000	add [bx+si],al
00007C3B	0000	add [bx+si],al
00007C3D	0000	add [bx+si],al
00007C3F	0000	add [bx+si],al
00007C41	0000	add [bx+si],al
00007C43	0000	add [bx+si],al
00007C45	0000	add [bx+si],al
00007C47	0000	add [bx+si],al
00007C49	0000	add [bx+si],al
00007C4B	0000	add [bx+si],al
00007C4D	0000	add [bx+si],al
00007C4F	0000	add [bx+si],al
00007C51	0000	add [bx+si],al
00007C53	0000	add [bx+si],al
00007C55	0000	add [bx+si],al
00007C57	0000	add [bx+si],al
00007C59	0000	add [bx+si],al
00007C5B	0000	add [bx+si],al
00007C5D	0000	add [bx+si],al
00007C5F	0000	add [bx+si],al
00007C61	0000	add [bx+si],al
00007C63	0000	add [bx+si],al

(2) 在启动过程，将 0x7a00 设置为断点进行调试



```

Next at t=0
(0) [0x0000ffffffffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b ; ea5be000
f0
<bochs:1> b 0x7c00
<bochs:2> c

```

```

(0) Breakpoint 1, 0x00007c00 in ?? ()
Next at t=14034561
(0) [0x000000007c00] 0000:7c00 (unk. ctxt): mov ax, cs ; 8cc8

```

```

<bochs:4> n
Next at t=14034562
(0) [0x000000007c02] 0000:7c02 (unk. ctxt): mov ds, ax ; 8ed8
<bochs:5> s
Next at t=14034563
(0) [0x000000007c04] 0000:7c04 (unk. ctxt): mov es, ax ; 8ec0

```

使用 n 和 s 进行单步运行

```

<bochs:3> info cpu
eax: 0x0000aa55 43605
ebx: 0x00000000 0
ecx: 0x00090000 589824
edx: 0x00000000 0
esp: 0x0000ffd6 65494
ebp: 0x00000000 0
esi: 0x000e0000 917504
edi: 0x0000ffac 65452
eip: 0x00007c00
eflags 0x00000082: id vip vif ac vm rf nt IOPL=0 of df if tf SF zf af pf cf
status word: 0x0000: b c3 TOS0 c2 c1 c0 es sf pe ue oe ze de ie
control word: 0x0040: inf RC_NEAREST PC_32 pm um om zm dm im
tag word: 0x5555
operand: 0x0000
fip: 0x00000000
fcs: 0x0000
fdp: 0x00000000
fds: 0x0000
=>FP0 ST0(0): raw 0x0000:0000000000000000 (0.0000000000) (ZERO)

```

使用 info cpu 查看 cpu 寄存器

## ② 程子洋:

- 1) 使用反汇编工具（例如 objdump）来反汇编生成的可执行文件

```

muye@muye-virtual-machine: ~/OS/osfs01-master
muye@muye-virtual-machine:~/OS/osfs01-master$ objdump -D -b binary -ml386 boot.bin

boot.bin: 文件格式 binary

Disassembly of section .data:

00000000 <.data>:
 0: 8c c8          mov     %cs,%eax
 2: 8e d8          mov     %eax,%ds
 4: 8e c0          mov     %eax,%es
 6: e8 02 00 eb fe call    0xfeeb000d
 b: b8 1e 7c 89 c5 mov     $0xc5897c1e,%eax
10: b9 0b 00 b8 01 mov     $0x1b8000b,%ecx
15: 13 bb 0c 00 b2 00 adc     0xb2000c(%ebx),%edi
1b: cd 10          int     $0x10
1d: c3            ret
1e: 43            inc     %ebx
1f: 48            dec     %eax
20: 45            inc     %ebp
21: 4e            dec     %esi
22: 47            inc     %edi
23: 5a            pop     %edx
24: 49            dec     %ecx
25: 59            pop     %ecx
26: 41            inc     %ecx
27: 4e            dec     %esi
28: 47            inc     %edi
..
1fd: 00 55 aa      add     %dl,-0x56(%ebp)

```

- 2) 使用命令 b 0x7c00 在 0x7c00 地址处设置断点，用命令 info break 展示





断点位置: 0x0000fffff6

查看所有断点: info break

```
00001052283i[BIOS] region 4: 0x0000c000
00001054341i[BIOS] PCI: bus=0 devfn=0x0b: vendor_id=0x8086 device_id=0x7113 cl
ass=0x0080
00001054617i[ACPI] new IRQ line = 11
00001054631i[ACPI] new IRQ line = 9
00001054659i[ACPI] new PM base address: 0xb000
00001054673i[ACPI] new SM base address: 0xb100
00001054701i[PCI] setting SMRAM control register to 0x4a
00001218836i[CPU0] Enter to System Management Mode
00001218847i[CPU0] RSM: Resuming from System Management Mode
00001382903i[PCI] setting SMRAM control register to 0xb0a
00001409519i[BIOS] MP table addr=0x000f9e70 MPC table addr=0x000f9da0 size=0xc
8
00001411226i[BIOS] SMBIOS table addr=0x000f9e80
00001413315i[BIOS] ACPI tables: RSDP addr=0x000f9fa0 ACPI DATA addr=0x01ff0000
size=0xff8
00001416299i[BIOS] Firmware waking vector 0x1ff0cc
00001418553i[PCI] i440FX PMC write to PAM register 59 (TLB Flush)
00001419283i[BIOS] bios_table_cur_addr: 0x000f9fc4
00001537403i[VBIOS] VGBios $Id: vgabios.c 288 2021-05-28 19:05:28Z vruppert $
00001537474i[BXVGA] VBE known Display Interface b0c0
00001537506i[BXVGA] VBE known Display Interface b0c5
00001540149i[VBIOS] VBE Bios $Id: vbe.c 292 2021-06-03 12:24:22Z vruppert $
00014034507i[BIOS] Booting from 0000:7c00

^C01619845902i[ ] Ctrl-C detected in signal handler.
Next at t=1619845903
(0) [0x000000007c09] 0000:7c09 (unk. ctxt): jmp .-2 (0x00007c09) ; ebf
<bochs:6> u
00007c09: ( ); jmp .-2 (0x00007c09) ; ebf
<bochs:7> info break
Num Type Disp Enb Address
1 pbreakpoint keep y 0x0000fffff6
<bochs:8>
```

#### 4. 为什么要 jmp \$, 如何改造程序, 让这个输出过程执行 100 次

##### (1) 原因

王卓:

jmp \$ 是一个无限循环的指令。\$ 表示当前指令的地址, 而 jmp \$ 意味着跳转到当前地址, 从而形成一个死循环。这是为了确保 CS:IP 持续停留在该地址, 而不继续向下执行。

程子洋:

jmp \$ 是一条汇编指令, 它实际上是一个无条件跳转指令, 将程序的控制流无条件地转移到当前指令的地址, 实际上是一个无限循环。在这个上下文中, jmp \$ 的作用是让程序无限循环, 不会退出, 以保持程序在执行状态, 否则程序执行完毕后将会退出, Bochs 模拟器也会停止运行。

聂森:

jmp \$ 在这里的作用是做一个无限循环, 让 CPU 停在此处, 不能再往下运行。假如没有这个无限循环, CPU 就会按着 CS:IP 一路往下执行, 能执行的执行, 不能执行的则 crash, 而我们不想看到这样的情况。

刘虢:

jmp \$ 指令用于在程序中创建一个无限循环。这意味着程序将会一直在该位置循环执行, 而不会继续向下执行。这通常在引导加载程序或需要持续运行而不退出的程序中使用。

##### (2) 改造程序

王卓:

要修改程序让输出过程执行 100 次, 可以引入一个计数器和一个条件跳转。

1. `org 07c00h` ; 告诉编译器程序加载到 7c00 处
2. `mov ax, cs`
3. `mov ds, ax`

```

4.  mov es, ax
5.  mov cx, 100 ; 将循环计数器设置为 100
6.
7.  Loop:
8.      push cx ; 保存 CX 的原始值
9.      call DispStr ; 调用显示字符串例程
10.     pop cx ; 恢复 CX 的原始值
11.     dec cx ; 将 CX 寄存器减一
12.     jnz Loop; 如果计数器不为零, 跳转到循环开始的地方
13.     jmp $ ; 无限循环
14.
15. DispStr:
16.     mov ax, BootMessage
17.     mov bp, ax ; ES:BP = 串地址
18.     mov cx, 16 ; CX = 串长度
19.     mov ax, 01301h ; AH = 13, AL = 01h
20.     mov bx, 000ch ; 页号为 0 (BH = 0) 黑底红字 (BL = 0Ch, 高亮)
21.     mov dl, 0
22.     int 10h ; 10h 号中断
23.     ret
24.
25. BootMessage: db "Hello, OS world!"
26.     times 510-($-$$) db 0 ; 填充剩下的空间, 使生成的二进制代码恰好为 512 字节
27.     dw 0xaa55 ; 结束标志

```

在修改后的代码中, 添加了一个计数器 `cx` 并使用栈来保存, 并在每次循环中使用 `dec` 指令将计数器减一。然后使用 `jnz` 指令检查计数器是否为零, 如果不为零, 跳转回循环开始处。这样循环执行 100 次后, 程序继续执行无限循环。这样, 输出过程将执行 100 次, 每次循环显示字符串 "Hello, OS world!"。

#### 程子洋:

- 添加了一个名为 `Counter` 的变量, 用于存储循环计数器的值, 初始化为 100;
- 在 `DispStr` 过程中, 添加了计数器的减 1 操作, 然后检查计数器是否为零, 如果不为零, 则跳转回 `DispStr` 重新显示字符串;
- 循环的次数由计数器控制, 这样程序将会输出字符串 100 次。

```

org 07c00h          ; where the code will be running
mov ax, cs
mov ds, ax
mov es, ax

; 初始化计数器, 放在0x8000处
mov word [Counter], 100

call DispStr
jmp $

DispStr:
mov ax, BootMessage
mov bp, ax          ; ES:BP = string address
mov cx, 16          ; CX = string length
mov ax, 01301h      ; AH = 13, AL = 01h
mov bx, 000ch       ; RED/BLACK
mov dl, 0
int 10h

; 计数器减1, 如果计数器不为0就跳转到DispStr重新显示
dec word [Counter]
jnz DispStr
ret

BootMessage: db "Hello, OS world!"
times 510-($-$$) db 0 ; fill zeros to make it exactly 512 bytes
dw 0xaa55

Counter: dw 0

```

**聂森:**

想要让这个输出过程执行 100 次, 可以在 `call DispStr` 前加上一句 `mov cx, 100`, 在后面加上一句 `loop $-2`, 即可实现 100 次循环。

**刘琥:**

```

org 07c00h

mov ax, cs
mov ds, ax
mov es, ax

mov cx, 100          ; Set loop counter to 100

PrintLoop:
call DispStr          ; Call the display string routine
loop PrintLoop        ; Decrement loop counter and loop until CX = 0

jmp $                ; Infinite loop

DispStr:
mov ax, BootMessage
mov bp, ax
mov cx, 16
mov ax, 01301h
mov bx, 000ch
mov dl, 0
int 10h
ret

BootMessage: db "Hello, OS world!"
times 510-($-$$) db 0
dw 0xaa55

```

## 5. 为什么要对段寄存器进行赋值

段寄存器赋值指令为:

1. `mov ax, cs`
2. `mov ds, ax`
3. `mov es, ax`

- 1) **内存分段机制：** x86 架构的计算机使用了一种内存分段机制，将物理内存分为多个段，每个段具有不同的起始地址和长度。段寄存器用于指定当前正在访问的内存段。
- 2) **代码段和数据段：** 通常情况下，x86 CPU 至少有两个重要的段寄存器，CS（代码段寄存器）和 DS（数据段寄存器）。CS 用于指定当前正在执行的代码所在的段，而 DS 用于指定数据的段。这种分离允许程序将代码和数据存储在不同的内存段中，以提高安全性和可维护性。
- 3) **内存保护：** 段寄存器的正确设置有助于实现内存保护。通过将不同的内存区域分配给不同的段，并设置相应的段寄存器，可以确保代码无法直接访问不属于它的数据，从而提高了系统的稳定性和安全性。
- 4) **多任务处理：** 在多任务处理环境中，不同的任务通常在不同的内存段中运行。通过更改段寄存器的值，操作系统可以切换任务，使每个任务都能够独立地访问自己的内存段，而不会干扰其他任务。
- 5) **内存映射：** 段寄存器的赋值还用于实现内存映射，这是一种将物理内存映射到虚拟内存地址的技术。不同的段寄存器值可以用于访问不同的内存映射区域。
- 6) **正确的内存访问：** 在 x86 架构的实模式下，内存访问是通过使用段寄存器和偏移地址的方式进行的。段寄存器存储了段选择子，而偏移地址则指定了相对于所选段的内存位置。段寄存器的默认值与代码段相等。因此，通过将 cs 的值复制给 ds 和 es，确保了代码段、数据段和附加段都指向同一个段，从而避免了出现段寄存器不一致的问题，确保正确的内存访问。

## 6. 如何在该程序中调用系统中断

在给定的代码中，通过以下指令调用了中断号为 10h 的系统中断：

```
1.  mov ax, 01301h ; AH = 13, AL = 01h
2.  mov bx, 000ch ; 页号为 0 (BH = 0) 黑底红字 (BL = 0Ch, 高亮)
3.  mov dl, 0
4.  int 10h ; 10h 号中断
```

在汇编语言中，要调用系统中断，可以使用 int 指令，该指令后面跟随一个中断号，表示要调用的系统中断的标识。不同的中断号对应不同的操作系统服务或功能。

这里使用 mov 指令将相应的值加载到寄存器 ax、bx 和 dl 中，然后使用 int 指令调用中断 10h。具体来说，这段代码调用了中断 10h，它是视频服务中断（BIOS 中断），用于在屏幕上显示文本。通过将适当的参数加载到寄存器中，然后调用中断 10h，进而实现在屏幕上显示指定的字符串。

## 三、实验过程分析

（实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等）

### 实验问题与故障分析：

王卓：

#### 1. 反汇编阅读命令无权限

```
wz@ubuntu:~/Desktop/chapter1/a$ sudo ndisasm -o 0x7c00 boot.bin >> disboot.asm
bash: disboot.asm: Permission denied
```

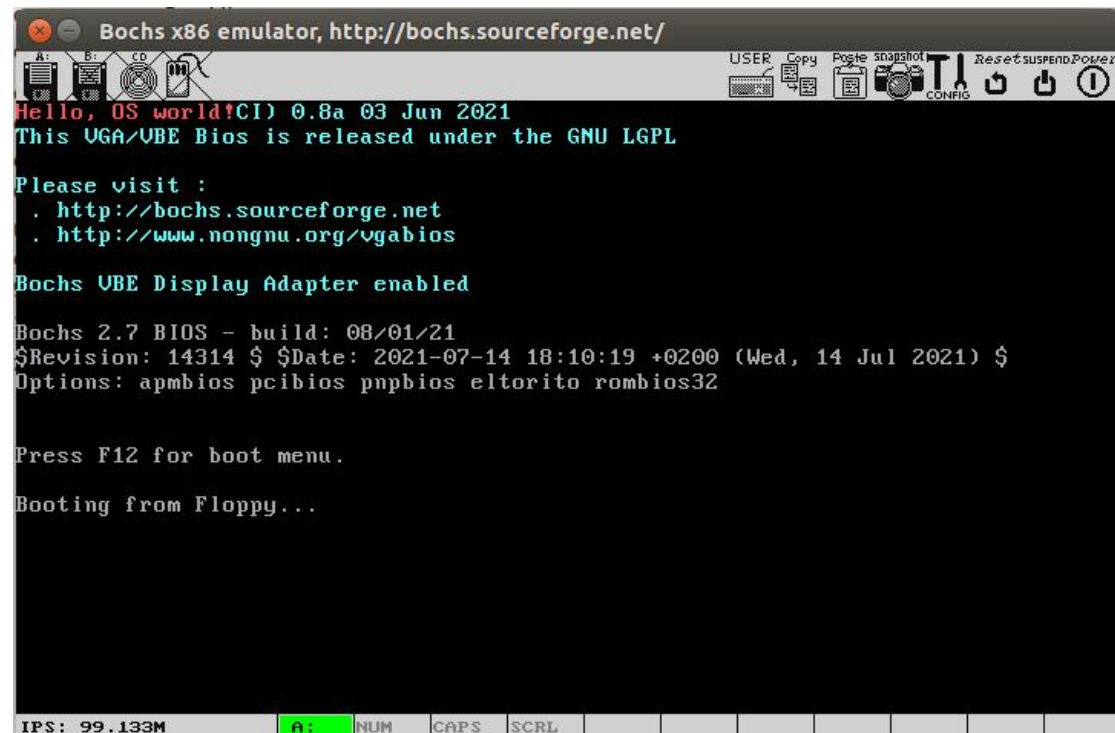
故障分析：

在实验过程中，修改 boot.asm 文件后无法保存说明其格式为只读，参考此问题，发现文件所在文件夹为只读格式，进而无法对编译后的文件进行操作  
故障排除方法：

1. `sudo chmod -R 777 chapter1/`

通过上述命令解除文件夹的只读格式

## 2. 删除 0xAA55 后程序仍成功运行

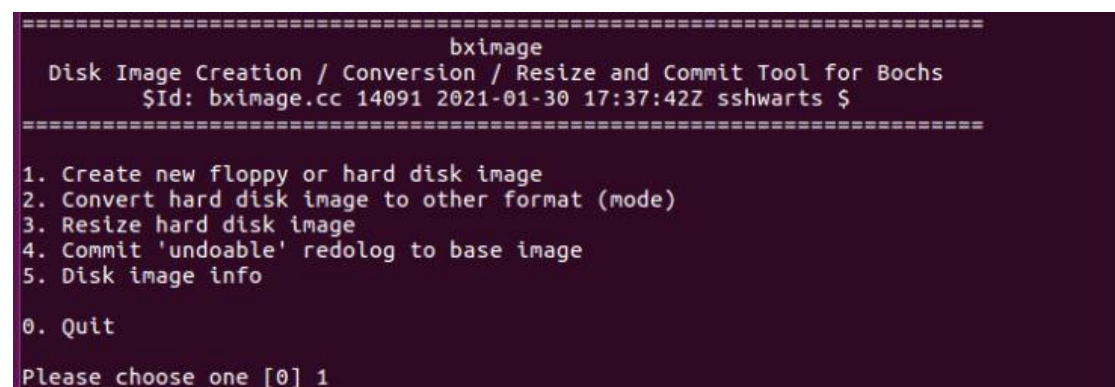


故障分析：

在再次创建虚拟软驱时，直接选择[0]导致直接 exit，使得后续写引导盘仍为能正常运行的可执行文件。

故障排除方法：

再次创建虚拟软驱时，选择[1]产生新的虚拟软驱覆盖之前的虚拟软驱



## 3. 无法在虚拟机和主机之间传递文件

解决方法：

1. 安装 VMware tools



2. 共享文件夹
3. 使用有关网站如奶牛快传，在主机中上传文件，在虚拟机中下载文件
4. 使用 u 盘，在主机中拷贝文件，在虚拟机中粘贴文件

程子洋：

### 1. 修改 boot.asm 文件后(即删除 0xAA55)后仍正常运行

故障分析：

使用了 MakeFile 文件进行整体编译，未注意 a.img 新的虚拟软驱的生成覆盖。即本质上仍使用了原有的 a.img。boot.asm 重新编译后的结果并未写入软驱。在使用 bxiimage 命令进行虚拟软驱的重新生成并覆盖后，完成了实验练习 1 的要求。

```
shizi@shizi-virtual-machine:~/osfs01-master$ bxiimage
=====
Disk Image Creation / Conversion / Resize and Commit Tool for Bochs
Sid: bxiimage.cc 14091 2021-01-30 17:37:42Z sshwartz S
=====
1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info
6. Quit
Please choose one [0] 1
Create image
Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd]
What kind of image should I create?
Please type flat, sparse, growing, vpc or vmware4. [flat]
Choose the size of hard disk sectors.
Please type 512, 1024 or 4096. [512]
Enter the hard disk size in megabytes, between 10 and 8257535
[10]
What should be the name of the image?
[c:\img] a.img
The disk image 'a.img' already exists. Are you sure you want to replace it?
Please type yes or no. [no] yes
Creating hard disk image 'a.img' with CHS=20/16/63 (sector size = 512)
The following line should appear in your bochsrc:
ata0-master: type=disk, path="a.img", mode=flat
=====
```

聂森：

### 1. 使用反汇编工具（例如 objdump）汇编文件时报错

故障分析：

```
osubuntu@ubuntu:~/Desktop/data/chapter1/a$ objdump -D -b -mi386 boot1.bin
objdump: boot1.bin: Invalid bfd target
osubuntu@ubuntu:~/Desktop/data/chapter1/a$ objdump -D -b binary boot1.bin
boot1.bin: file format binary
objdump: can't disassemble for architecture UNKNOWN!
```

方法一：查找指令格式，正确书写指令：objdump -D -b binary -mi386 boot.bin，其中 objdump 是一个用于分析二进制文件的工具，-D 选项告诉 objdump 要反汇编 (disassemble) 文件，-b binary 选项指定输入文件的格式为二进制文件，-mi386 选项指定了目标架构为 Intel x86 (i386)。

方法二：ndisasm -b 16 boot2.bin，其中 ndisasm 是一个用于反汇编 x86 汇编语言的工具，-b 16 选项指定了目标架构为 16 位 x86，boot2.bin 是要反汇编的二进制文件。

刘虢：

### 1. 无法打开文件 boot2.asm

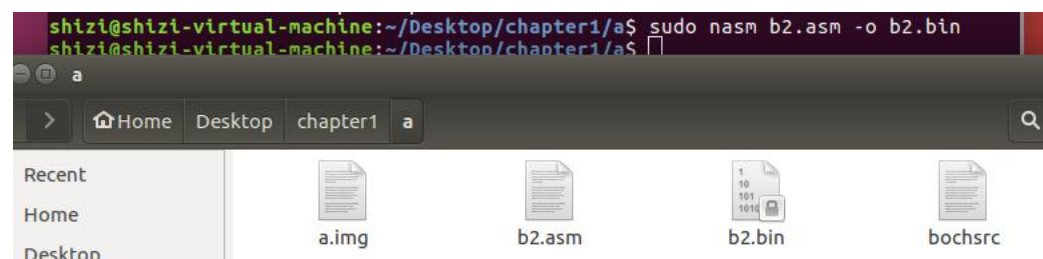
```
shizi@shizi-virtual-machine:~/Desktop/chapter1/a$ sudo nasm boot2.asm -o boot2.bin
[sudo] password for shizi:
nasm: fatal: unable to open input file 'boot2.asm'
shizi@shizi-virtual-machine:~/Desktop/chapter1/a$
```

故障分析：

dos 系统不支持 8 位以上的文件名

解决方案：

将文件名改为 8 位或 8 位以下



## 2. 无法在虚拟机和主机之间传递文件

解决方法：

1. 安装 VMware tools
2. 共享文件夹

## 四、实验结果总结

（对实验结果进行分析，完成思考题目，并提出实验的改进意见）

### 实验结果总结：

1. 实验目标达成情况：成功地搭建了实验环境，安装了 32 位 Ubuntu16.04，编译并安装了 Bochs2.7。
2. 问题和困难：在实验中，遇到了一些问题，如 Bochs 的编译配置、源码位置修改，虚拟软驱未覆盖等问题。这些问题都实现了解决。
3. 实验数据和观察：使用 Bochs 运行第一个引导程序。成功运行的程序输出了 "Hello, OS world!"。
4. 实验经验：通过这次实验，我们获得了搭建实验环境、编译和运行 Bochs 的经验。这对于操作系统开发的进一步学习是有益的。

### 实验改进意见：

#### 王卓：

此次实验为搭建实验环境并熟悉开发与调试工具，是后续操作系统实验的基础。但除了完成相关实验内容外，个人认为还能做一些更全面的实验探索，比如尝试修改引导程序的其他部分，并观察对应的结果和行为的变化；了解一下引导扇区的结构和启动过程等内容，进而更全面地了解操作系统的引导过程。

#### 程子洋：

1. 扩展性实验：逐渐增加实验的复杂性，引入更多的挑战和新概念，以便能够逐渐提高技能并深入了解操作系统开发。
2. 深化理论知识：在实验之前，提供相关的理论知识背景，使学生了解操作系统的基本概念、原理和关键组成部分。这将帮助他们更好地理解实验的上下文和目标。
3. 安全性考虑：引入关于操作系统安全性的知识，以理解和应用安全性原则和最佳实践。
4. 代码分析：希望可以深入分析操作系统内核或引导程序的源代码，以便更好

地理解代码结构和工作原理。加强编程技能和代码理解能力。

**刘琥：**

本次实验推荐使用的 virtualbox 软件过于笨重，可以改用更为方便的 VMware Workstation 平台来搭建虚拟机，在操作时和运行时可以更加方便。

## 五、各人实验贡献与体会（每人各自撰写）

**同学：王卓**

此次实验为本人独立完成全部实验内容，撰写一份实验报告并对小组成员实验报告进行整合，这种独立完成实验的过程给我的体会如下：

1. 通过个人独立完成实验内容，锻炼了我解决问题的能力。在实验过程中，遇到了如反汇编命令无权限等问题，通过主动寻找解决方案并成功克服了困难。
2. 通过个人完成全部实验内容，我对安装虚拟运行环境、搭建开发环境、编译和调试程序等步骤有了更深入的理解。我不仅仅是按照指导完成实验，更是通过实际操作和自己的思考，对操作系统实验的流程和原理有了更深入的认识。
3. 个人独立完成实验内容，需要我主动查阅资料、解决问题和探索实验的不同方面。这培养了我的自主学习能力，能够独立掌握新知识和技能，不依赖他人的指导。
4. 通过实际操作，将所学的理论知识如汇编语言应用到实验问题的解决中，加深了对操作系统原理和概念的理解。实践与理论相结合，帮助我更好地理解 and 消化学术知识。

**同学：程子洋**

独立完成全部实验与撰写一份实验报告

实验体会：

1. 环境配置的挑战：一开始，搭建实验环境和配置 Bochs 等工具似乎是一项艰巨的任务。需要下载和安装多个软件包，还需要对系统环境进行配置。这使我更加了解了操作系统开发背后的底层知识，包括编译工具和库的安装以及系统配置。
2. 汇编语言编程：在实验中，我复习了 x86 汇编代码的内容，复习了这种底层的编程语言。通过编写和调试汇编代码，我更深入地理解了计算机的工作原理，包括寄存器、内存访问和指令执行。
3. 调试技巧的提高：遇到问题时，我不得不运用各种调试技巧来识别和解决错误。使用 Bochs 的调试器，我学会了设置断点、查看寄存器内容、反汇编代码等技能，这对于理解程序的执行过程和定位问题非常有帮助。
4. 实验的逐步完成：实验按照逐步的方式进行，每个步骤都构建在前一个步骤的基础上。这种方法使我能够逐渐增加对操作系统引导过程的理解。
5. 操作系统启动流程的理解：完成实验后，我对计算机启动操作系统的过程有了更清晰的认识。我了解了 BIOS 启动、引导扇区加载和操作系统启动的关键步骤。这为我提供了对计算机内部工作的宝贵见解。

**同学：聂森**

独立完成全部实验并参与撰写实验报告

实验体会：

通过本次实验，我学会了使用 bochs 模拟器来运行汇编程序、使用 bximage 工具来创建虚拟软驱、编写汇编程序和反汇编汇编程序。通过解决实验中遇到的问题，如如何使用反汇编程序等，我对汇编语言和操作系统的运行有了更深入的了解。

<p>在实验时，我发现了自己的动手实践能力急需提高，原以为一些已经掌握的东西在实验过程中发现自己并没有掌握，一些以为不会卡住的环节做的时候发现原来自己不会做。纸上得来终觉浅，绝知此事要躬行。在日后学习中，我需要去落实基本知识、更多锻炼实验能力。</p> <p><b>同学：刘琥</b></p> <p>独立完成环境搭建与绝大部分实验内容，最后学习组内其他成员的经验成果完成反汇编和断点调试。</p> <p>本次实验让我对虚拟机的使用更加熟悉，锻炼了独立解决问题的能力，在遇到虚拟机报错时能条理清晰地找到问题所在并知道如何找到解决方案。同时课内的知识的实际运用让我对知识的掌握更加牢固，面对汇编码更加得心应手。</p>		
<b>六、教师评语</b>		
<b>教师评分（请填写好姓名、学号）</b>		
姓名	学号	分数
王卓	2021302191791	
程子洋	2021301051114	
聂森	2021302191536	
刘琥	2021302121234	
教师签名：		
年 月 日		