

□ 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	2024.9.14
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
黄东威	2022302181148	信息安全	5班
王浚杰	2022302181143	网络空间安全	5班
程序	2022302181131	信息安全	4班
周业营	2022302181145	信息安全	5班

一、实验目的及实验内容

(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)

(实验目的、内容及相关的理论知识)

实验目的

- 搭建基本实验环境
- 熟悉基本开发与调试工具

实验内容

- 在实验机上安装虚拟运行环境，并安装Ubuntu
- 安装Ubuntu开发环境，32位环境
- 下载Bochs源码，编译并安装Bochs环境
- 使用Bochs自带工具 `bximage` 创建虚拟软驱
- 阅读、编译 `boot.asm`，并反汇编阅读
- 修改 `bochssrc`，运行并调试第一个程序
- 完成实验练习要求

原理分析

Bochs的核心功能是模拟一台完整的x86计算机，因此它的使用需要配置好虚拟硬件和启动环境。实验中，我们主要使用Bochs来模拟引导扇区的运行，观察操作系统引导程序的执行。以下是其原理分析和启动过程的步骤：

原理分析

- **Bochs配置文件 bochsrc**: Bochs的运行依赖一个配置文件（`bochsrc`），用于定义虚拟机的硬件环境，包括CPU型号、内存大小、显示器类型、虚拟软驱和硬盘位置等。通过修改该文件，我们可以自定义虚拟硬件的配置，确保实验程序能够在模拟环境中正常运行。
- 例如，在`bochsrc`中，我们可以指定虚拟软驱和硬盘的映像文件路径，以及屏幕输出使用的图形库（如`sdl`或`x11`）。
- **虚拟软驱与硬盘的创建**: Bochs自带的工具`bximage`用于创建虚拟软盘或硬盘镜像。虚拟软盘通常用来加载引导扇区，模拟启动设备。通过`bximage`生成的软盘映像文件可以作为虚拟软驱，加载实验程序的引导代码。
- **Bochs调试工具**: Bochs内置了强大的调试功能，能够通过命令行界面执行单步调试、设置断点、反汇编代码等。在本实验中，涉及到的调试器功能如下：
 - **断点设置**: 通过`b address`设置断点，暂停程序在指定地址的执行。
 - **单步执行**: 通过`s`或`n`指令逐条执行汇编指令，观察程序的执行过程。`s`指令可以跳入函数内部，`n`指令可以跳过函数内部。
 - **寄存器查看**: 通过`r`命令可以实时查看通用寄存器和控制寄存器的值，帮助分析程序状态。
 - **内存查看**: 通过`x /nuf address`命令可以查看线性地址内容。其中，/nuf分为两部分：需要显示的字节数以及输出格式的指示符。例如，`x /64xb 0x7c00`表示从`0x7c00`开始显示64字节的地址内容，格式为16进制，以字节为单位显示。
 - **反汇编**: 通过`u 起始地址 终止地址`命令可以对生成的可执行文件进行反汇编，查看其汇编代码。
- **引导程序的运行与调试**: 在Bochs中，我们可以加载编译好的引导程序（通常是汇编语言写的`boot.asm`），模拟BIOS启动过程，并在虚拟机中执行此引导程序。通过Bochs调试器，可以实时监控引导程序的运行，分析输出结果，帮助理解操作系统从加电启动到内核加载的全过程。

启动步骤

Bochs模拟的x86启动过程与真实计算机的启动过程相同，涉及到以下步骤：

1. **BIOS初始化**: 当Bochs启动时，它会模拟x86处理器的启动状态，执行BIOS引导代码，进行硬件初始化和自检。
2. **加载引导扇区**: BIOS在初始化完成后，会从指定的启动设备（如虚拟软盘或硬盘）加载引导扇区，将其复制到内存中的特定位置（如`0x7c00`），并将控制权交给引导程序。
3. **执行引导程序**: 引导程序负责进一步初始化操作系统，通常包括加载操作系统内核，将控制权传递给内核等任务。

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；具体实验步骤)

实验环境

- **虚拟化软件**: VMware Workstation Pro

VMware Workstation Pro 用于在宿主机上运行虚拟机。本次实验使用VMware创建并运行 Kubuntu 18.04.5 32位系统的虚拟机，便于实验操作环境的搭建与调试。

- **操作系统**: Kubuntu 18.04.5 32位版本

本次实验使用的虚拟机操作系統是Kubuntu 18.04.5 , 32位版本。Kubuntu是Ubuntu的官方派生版本，采用KDE Plasma桌面环境，在操作体验上更为轻量和美观。

实验步骤

安装环境

1. 安装VMware, 添加系统镜像, 创建并运行Kubuntu 18.04.5 32位系统的虚拟机
2. 安装开发环境和依赖库

```
sudo apt-get install libSDL2-dev
sudo apt-get install vim
sudo apt-get install build-essential
sudo apt-get install xorg-dev
sudo apt-get install libgtk2.0-dev
sudo apt-get install g++
```

3. 从[Bochs 下载页面](#)下载Bochs2.7源码包

4. 解压Bochs源码包

```
tar -zxvf bochs-2.7.tar.gz
```

```
file(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
rimeheart@Rimeheart : ~/oslab/lab0 $ tar -zxvf bochs-2.7.tar.gz
bochs-2.7/
bochs-2.7/bios/
bochs-2.7/bios/VGABIOS-elpin-LICENSE
bochs-2.7/bios/notes
bochs-2.7/bios/BIOS-bochs-latest
bochs-2.7/bios/BIOS-bochs-legacy
bochs-2.7/bios/rombios.c
bochs-2.7/bios/rombios.h
bochs-2.7/bios/rombios32.c
bochs-2.7/bios/VGABIOS-lgpl-README
bochs-2.7/bios/VGABIOS-lgpl-latest
bochs-2.7/bios/VGABIOS-lgpl-latest-banshee
bochs-2.7/bios/VGABIOS-lgpl-latest-cirrus
bochs-2.7/bios/VGABIOS-lgpl-latest-cirrus-debug
bochs-2.7/bios/VGABIOS-lgpl-latest-debug
bochs-2.7/bios/Makefile.in
bochs-2.7/bios/rombios32start.S
bochs-2.7/bios/biossums.c
bochs-2.7/bios/rombios32.ld
bochs-2.7/bios/SeaBIOS-README
bochs-2.7/bios/SeaVGABIOS-README
bochs-2.7/bios/bios.bin-1.13.0
```

5. 进入Bochs目录并进行配置:

```
cd bochs-2.7/
./configure --prefix=/home/rimeheart/oslab/lab0 --enable-debugger-gui --
enable-disasm --enable-iodebug --enable-x86-debugger --with-x --with-x11 --
with-sdl --enable-debugger
```

```
bochs-2.7 : bash — Konsole
文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
rimeheart@Rimeheart : ~/oslab/lab0/bochs-2.7      $ ./configure --prefix=/home/rimeheart/oslab
/bochs-2.7 --enable-debugger-gui --enable-disasm --enable-iodebbug --enable-x86-debugger --w
ith-x --with-x11 --with-sdl --enable-debugger
configure: WARNING: unrecognized options: --enable-disasm
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking if you are configuring for another platform... no
checking for standard CFLAGS on this platform...
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking whether make sets $(MAKE)... yes
checking for a sed that does not truncate output... /bin/sed
```

6. 编译并安装Bochs:

```
make
make install
```

```
bochs-2.7 : make — Konsole
文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
rimeheart@Rimeheart : ~/oslab/lab0/bochs-2.7      $ make
cd iodev && \
make libiodev.a
make[1]: 进入目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev"
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread devices.cc -o devices.o
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread virt_timer.cc -o virt_timer.o
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread slowdown_timer.cc -o slowdown_timer.o
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread pic.cc -o pic.o
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread pit.cc -o pit.o
g++ -c -I.. -I./.. -I./instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE_OFFSET_BITS=64 -D_LARGE_FILES -pthread serial.cc -o serial.o
serial.cc: In static member function 'static void bx_serial_c::tx_timer()':
serial.cc:1454:14: warning: ignoring return value of 'ssize_t write(int, const void*, size_t)', declared with attribute warn_unused_result [-Wunused-result]
    write(BX_SER_THIS s[port].tty_id, (bx_ptr_t) & BX_SER_THIS s[port].tsrbuffer,
  1);
```

```
rimeheart@Rimeheart : ~/oslab/lab0/bochs-2.7      $ make install
cd iodev && \
make libiodev.a
make[1]: 进入目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev"
make[1]: "libiodev.a"    已是最新。
make[1]: 离开目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev"
echo done
done
cd iodev/display && \
make libdisplay.a
make[1]: 进入目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev/display"
make[1]: "libdisplay.a"    已是最新。
make[1]: 离开目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev/display"
echo done
done
cd iodev/hdimage && \
make libhdimage.a
make[1]: 进入目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev/hdimage"
make[1]: "libhdimage.a"    已是最新。
make[1]: 离开目录 "/home/rimeheart/oslab/lab0/bochs-2.7/iodev/hdimage"
echo done
done
cd bx_debug && \

```

Bochs使用

1. bochsrc 配置文件的使用

在项目目录下创建一个文件，叫做 `bochsrc`，该文件用于定义虚拟机的硬件参数。Bochs 需要依赖此文件来加载BIOS、磁盘镜像等资源。

```
touch bochsrc
vim bochsrc
```

添加以下内容：

```
# 指定虚拟机的BIOS镜像
romimage: file=/home/rimeheart/oslab/lab0/share/bochs/BIOS-bochs-latest
vgaromimage: file=/home/rimeheart/oslab/lab0/share/bochs/VGABIOS-lgpl-latest

# keyboard:keymap=/home/rimeheart/oslab/lab0/share/bochs/keymaps/x11-pc-
us.map

# 1.44=磁盘镜像位置
floppya: 1_44="boot.img", status=inserted

# 从软盘启动
boot: floppy

display_library: sdl

# 置鼠标不可用
mouse: enabled=0
```

```
megs: 512
```

```
文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
# 指定虚拟机的BIOS镜像
romimage: file=/home/rimeheart/oslab/lab0/share/bochs/BIOS-bochs-latest
vgaromimage: file=/home/rimeheart/oslab/lab0/share/bochs/VGABIOS-lgpl-latest

# keyboard:keymap=/home/rimeheart/oslab/lab0/share/bochs/keymaps/x11-pc-us.map

# 1.44= 磁盘镜像位置
floppya: 1_44="boot.img", status=inserted

# 从软盘启动
boot: floppy

display_library: sdl

# 置鼠标不可用
mouse: enabled=0

megs: 512

-- 插入 --          20,1          全部
[bin : vim]
```

补充：也可通过参考教材配套的github仓库下载源码，将vgaromimage对应的文件位置修改为自己实际的位置，注释掉keyboard_mapping一行，并增加display_library: sdl

```
git clone https://github.com/yyu/osfs02.git
```

2. 创建引导扇区程序

```
touch boot.asm
vim boot.asm
```

程序如下，用于显示"Hello, OS world!"字符串，之后进入无限循环。代码包含必要的BIOS中断调用，并确保引导扇区大小为512字节，其中最后两个字节是 0xAA55 的引导扇区签名。

```
org 07c00h           ; where the code will be running
mov ax, cs
mov ds, ax
mov es, ax
call DispStr         ; let's display a string
jmp $                ; and loop forever
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
    ret
BootMessage: db "Hello, OS world!"
times 510-($-$) db 0   ; fill zeros to make it exactly 512 bytes
```

```
dw 0xaa55 ; boot record signature
```

The screenshot shows a terminal window titled "bin : vim — Konsole". The assembly code is as follows:

```
org 0 7c00h ; where the code will be running
mov ax, cs
mov ds, ax
mov es, ax
call DispStr ; let's display a string
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 ; BX = video segment
    mov dl, 0
    int 10h
    ret

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

~  
~  
~  
:wq
```

3. 编译引导扇区程序

使用 `nasm` 将 `boot.asm` 汇编成二进制文件 `boot.bin`:

```
nasm boot.asm -o boot.bin
```

4. 创建虚拟软盘镜像

使用 `bximage` 工具创建一个虚拟软盘镜像 `boot.img`, 用于模拟软盘引导。此命令创建一个大小为1.44MB的软盘镜像 `boot.img`, 供虚拟机启动时使用。

```
bximage -mode=create -fd=1.44M -q boot.img
```

此命令创建一个大小为1.44MB的软盘镜像 `boot.img`, 供虚拟机启动时使用。

```
bin : bash — Konsole
文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
rimeheart@Rimeheart : ~/oslab/lab0/bin      $ bximage
=====
bximage
Disk Image Creation Tool for Bochs
$Id: bximage.c 11315 2012-08-05 18:13:38Z vruppert $

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, in megabytes.
Please type 0.16, 0.18, 0.32, 0.36, 0.72, 1.2, 1.44, 1.68, 1.72, or 2.88.
[1.44] 1.44
I will create a floppy image with
cyl=80
heads=2
sectors per track=18
total sectors=2880
total bytes=1474560

What should I name the image?
[a.img] boot.img

bin : bash
```

5. 将引导程序写入软盘镜像

使用 `dd` 工具将编译好的引导程序 `boot.bin` 写入 `boot.img` 中, 确保程序加载到软盘的第一扇区。

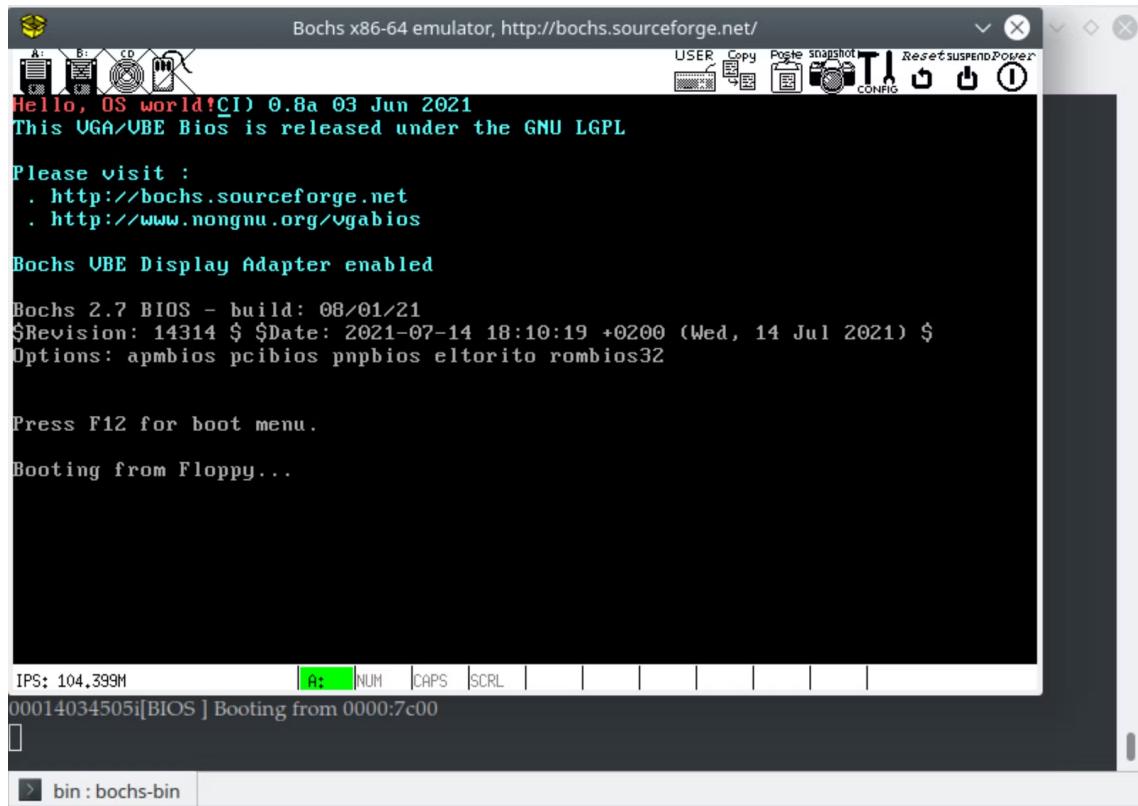
```
dd if=boot.bin of=boot.img bs=512 count=1 conv=notrunc
```

```
rimeheart@Rimeheart : ~/oslab/lab0/bin      $ dd if=boot.bin of=boot.img bs=512 count=1 conv=n
otrunc
记录了 1+0 的读入
记录了 1+0 的写出
512 bytes copied, 0.000174212 s, 2.9 MB/s
```

6. 模拟系统环境启动

```
bochs -f bochssrc
```

成功输出字符串!



实验练习

1. 删除0xAA55，观察程序效果，找出原因

0xAA55 是引导扇区的标志位，通常位于引导扇区的最后两个字节。BIOS 通过检查这两个字节来判断该扇区是否包含有效的引导程序。如果删除这两个字节，程序将无法通过 BIOS 的引导扇区校验，从而无法启动操作系统，出现 PANIC

A screenshot of a terminal window titled "bin : bochs-bin — Konsole". The window contains the following text:

```
000014034505i[BIOS ] Booting from 0000:7c00
000014034505i[BIOS ] ACPI tables: RSDP addr=0x000f9fa0 ACPI DATA addr=0x1fff0000 size=0
00001413390i[BIOS ] ACPI tables: RSDP addr=0x000f9fa0 ACPI DATA addr=0x1fff0000 size=0
00001416397i[BIOS ] Firmware waking vector 0x1fff00cc
00001418651i[PCI ] 440FX PMC write to PAM register 59 (TLB Flush)
00001419381i[BIOS ] bios_table_cur_addr: 0x000f9fc4
00001537501i[VBIOS] VGA Bios $Id: vgabios.c 288 2021-05-28 19:05:28Z vruppert $
00001537572i[BXVGA] VBE known Display Interface b0c0
00001537604i[BXVGA] VBE known Display Interface b0c5
00001540247i[VBIOS] VBE Bios $Id: vbe.c 292 2021-06-03 12:24:22Z vruppert $
00014065603p[BIOS ] >>PANIC<< No bootable device.

=====
Event type: PANIC
Device: [BIOS ]
Message: No bootable device.

A PANIC has occurred. Do you want to:
cont - continue execution
alwayscont - continue execution, and don't ask again.
    This affects only PANIC events from device [BIOS ]
die - stop execution now
abort - dump core
debug - continue and return to bochs debugger
Choose one of the actions above: [die]
```

The status bar at the bottom shows "bin : bochs-bin".

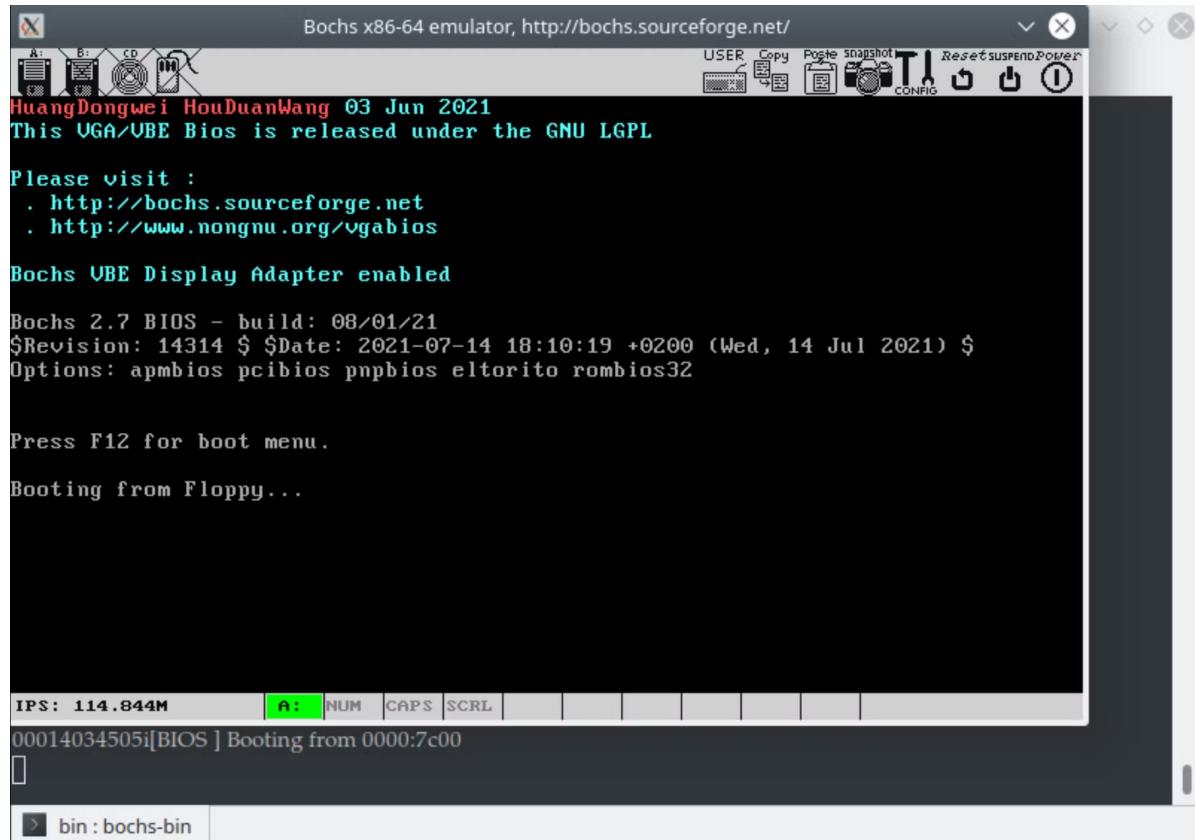
2. 修改程序中输出为，一个包含自己名字的字符串，调试程序

修改汇编程序中的字符串，将其替换为包含自己名字的内容，并观察输出是否正确。

```
BootMessage: db "Hello, OS world!" ;修改前  
BootMessage: db "HuangDongwei HouDuanWang" ;修改为名字
```

字符串长度超过了 CX 中指定的长度，需要进行修改：

```
mov cx, 16          ; 修改前  
mov cx, 24          ; 修改后 CX = string length
```

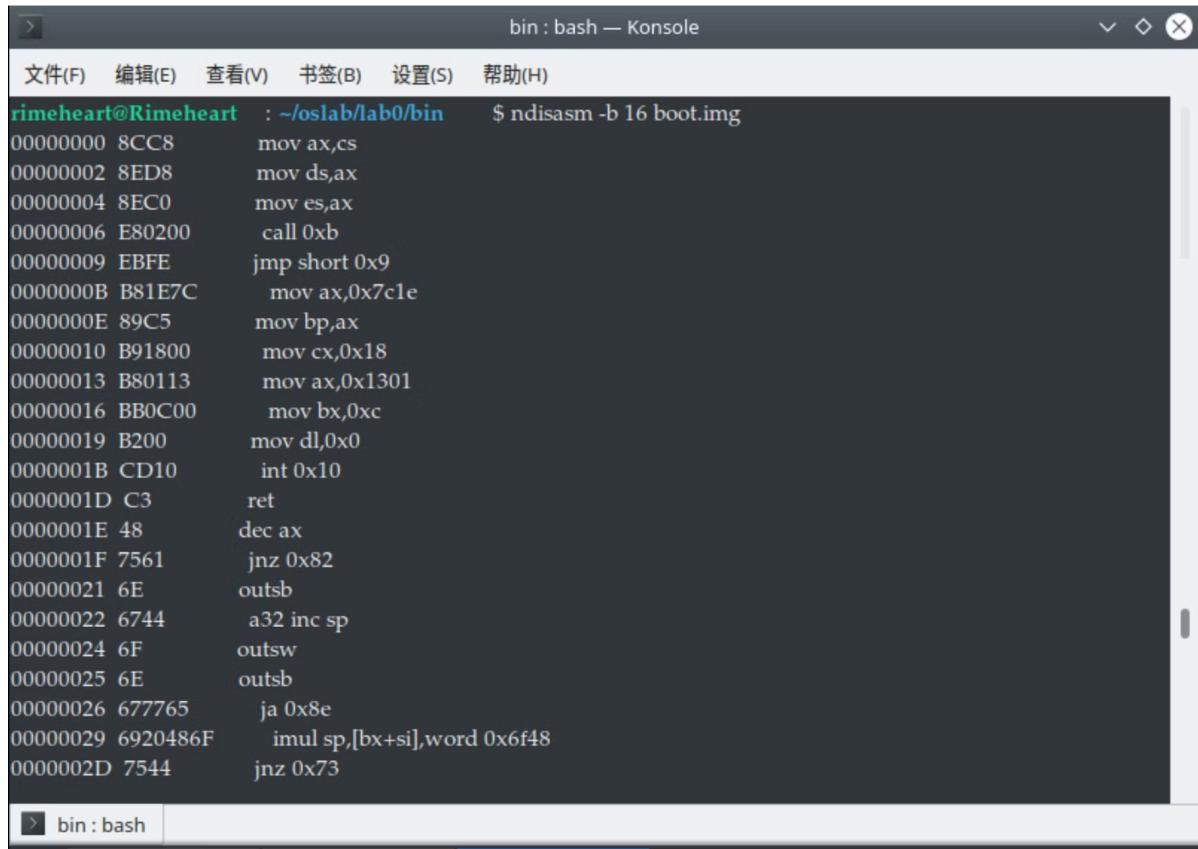


3. 把生成的可执行文件反汇编，查看输出内容，设置断点调试

通过 `nasm` 汇编后生成的 `boot.bin` 文件，可以使用 Bochs 的调试器功能反汇编查看程序内容，并在程序的关键位置设置断点进行调试。

```
ndisasm -b 16 boot.img
```

可以看到反汇编结果能与 `nasm` —— 对应



bin : bash — Konsole

文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)

```
rimeheart@Rimeheart : ~/oslab/lab0/bin      $ ndisasm -b 16 boot.img
00000000 8CC8      mov ax,cs
00000002 8ED8      mov ds,ax
00000004 8EC0      mov es,ax
00000006 E80200    call 0xb
00000009 EBFE      jmp short 0x9
0000000B B81E7C    mov ax,0x7c1e
0000000E 89C5      mov bp,ax
00000010 B91800    mov cx,0x18
00000013 B80113    mov ax,0x1301
00000016 BB0C00    mov bx,0xc
00000019 B200      mov dl,0x0
0000001B CD10      int 0x10
0000001D C3        ret
0000001E 48        dec ax
0000001F 7561      jnz 0x82
00000021 6E        outs
00000022 6744      a32 inc sp
00000024 6F        outsw
00000025 6E        outs
00000026 677765    ja 0x8e
00000029 6920486F   imul sp,[bx+si],word 0x6f48
0000002D 7544      jnz 0x73
```

之后，在 bochsrc 中加入启动调试的配置

```
magic_break: enabled=1          # 启用调试器
debug: action=ignore
```

在引导扇区 0x7c00 开始处设置断点，执行至此位置

```
b 0x7c00      # 在引导扇区开始处设置断点
c
```

此时 BIOS 加载引导扇区的地址，执行 boot.asm 中的指令

```
bin : bochs-bin — Konsole
文件(F) 编辑(E) 查看(V) 书签(B) 设置(S) 帮助(H)
(0) [0x0000000000007c00] 0000:7c00 (unk. ctxt): mov ax, cs ; 8cc8
<bochs:3> s
Next at t=14034561
(0) [0x0000000000007c02] 0000:7c02 (unk. ctxt): mov ds, ax ; 8ed8
<bochs:4> s
Next at t=14034562
(0) [0x0000000000007c04] 0000:7c04 (unk. ctxt): mov es, ax ; 8ec0
<bochs:5> s
Next at t=14034563
(0) [0x0000000000007c06] 0000:7c06 (unk. ctxt): call .+2 (0x00007c0b) ; e80200
<bochs:6> s
Next at t=14034564
(0) [0x0000000000007c0b] 0000:7c0b (unk. ctxt): mov ax, 0x7c1e ; b81e7c
<bochs:7> s
Next at t=14034565
(0) [0x0000000000007c0e] 0000:7c0e (unk. ctxt): mov bp, ax ; 89c5
<bochs:8> s
Next at t=14034566
(0) [0x0000000000007c10] 0000:7c10 (unk. ctxt): mov cx, 0x0018 ; b91800
<bochs:9> s
Next at t=14034567
(0) [0x0000000000007c13] 0000:7c13 (unk. ctxt): mov ax, 0x1301 ; b80113
<bochs:10> s
```

再设置断点至 `int 10h` 前，执行这个中断后，将会将修改后的字符串打印

```
b 0x7c1b
c
n
```

```
Bochs x86-64 emulator, http://bochs.sourceforge.net/
Bochs UGABios (PCI) 0.8a 03 Jun 2021
This UGA/VBE Bios is released under the GNU LGPL

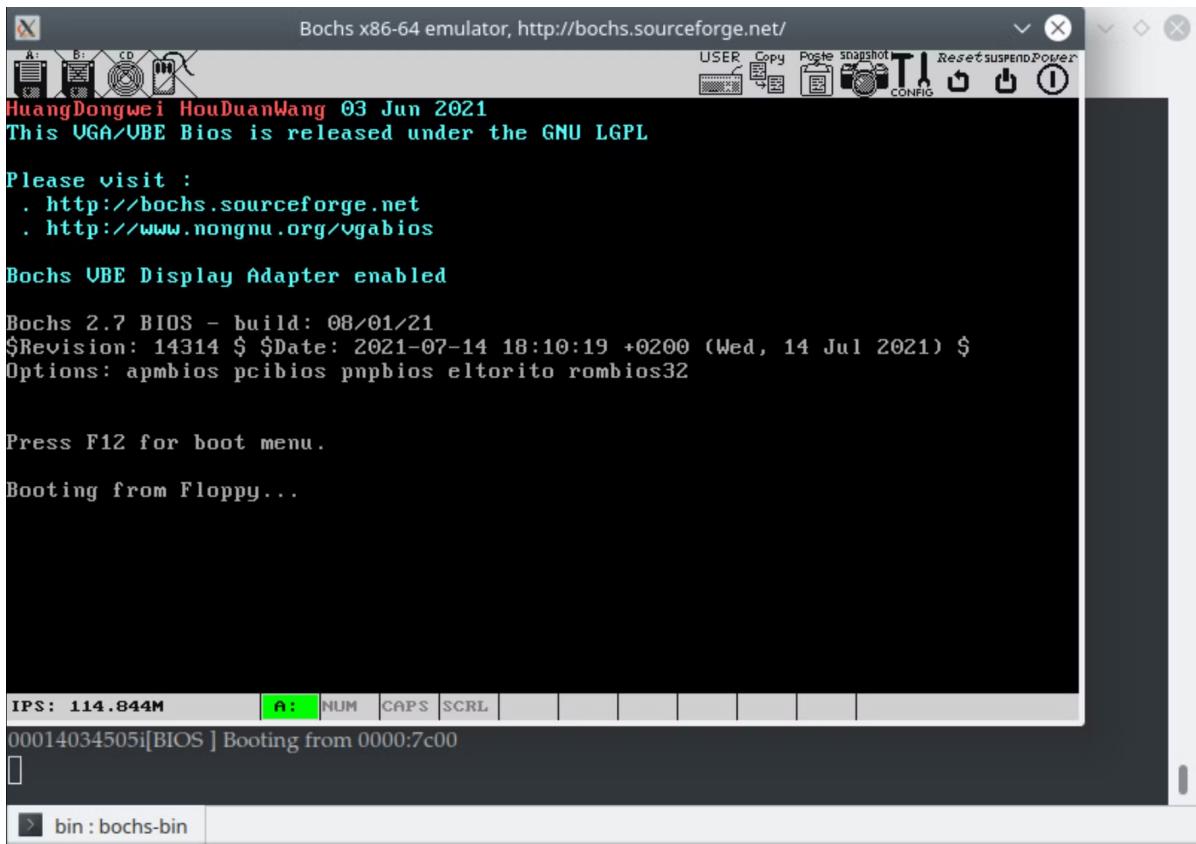
Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

Bochs VBE Display Adapter enabled

Bochs 2.7 BIOS - build: 08/01/21
$Revision: 14314 $ $Date: 2021-07-14 18:10:19 +0200 (Wed, 14 Jul 2021) $
Options: apmbios pcibios pnpbios eltorito rombios32

Press F12 for boot menu.

Booting from Floppy...
A: NUM CAPS SCRL
(0) [0x0000000000007c1b] 0000:7c1b (unk. ctxt): int 0x10 ; cd10
<bochs:3>
```



补充：也可以通过bochs的u指令来完成反汇编。先打开bochs，使得boot.bin被加载进内存，通过x \nuf 0x7c00来查看内存内容，确定程序的终止地址为0x7dff，然后使用u 0x7c00 0x7e00来对该段可执行程序反汇编，实验结果如下：

```
<bochs:17> x /256 0x7d00
[bochs]:
0x000007d00 <bogus+    0>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d08 <bogus+    8>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d10 <bogus+   16>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d18 <bogus+   24>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d20 <bogus+   32>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d28 <bogus+   40>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d30 <bogus+   48>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d38 <bogus+   56>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d40 <bogus+   64>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d48 <bogus+   72>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d50 <bogus+   80>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d58 <bogus+   88>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d60 <bogus+   96>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d68 <bogus+  104>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d70 <bogus+  112>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d78 <bogus+  120>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d80 <bogus+  128>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d88 <bogus+  136>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d90 <bogus+  144>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007d98 <bogus+  152>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007da0 <bogus+  160>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007da8 <bogus+  168>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007db0 <bogus+  176>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007db8 <bogus+  184>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007dc0 <bogus+  192>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007dc8 <bogus+  200>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007dd0 <bogus+  208>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007dd8 <bogus+  216>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007de0 <bogus+  224>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007de8 <bogus+  232>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007df0 <bogus+  240>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x000007df8 <bogus+  248>: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x55 0xaa
<bochs:18>
```

```

<bochs:20> u 0x7c00 0x7e00
00007c00: (          ): mov ax, cs           ; 8cc8
00007c02: (          ): mov ds, ax           ; 8ed8
00007c04: (          ): mov es, ax           ; 8ec0
00007c06: (          ): call .+2  (0x00007c0b) ; e80200
00007c09: (          ): jmp .-2  (0x00007c09) ; ebfe
00007c0b: (          ): mov ax, 0x7c1e        ; b81e7c
00007c0e: (          ): mov bp, ax           ; 89c5
00007c10: (          ): mov cx, 0x0010        ; b91000
00007c13: (          ): mov ax, 0x1301        ; b80113
00007c16: (          ): mov bx, 0x000c        ; bb0c00
00007c19: (          ): mov dl, 0x00           ; b200
00007c1b: (          ): int 0x10           ; cd10
00007c1d: (          ): ret                 ; c3
00007c1e: (          ): dec ax             ; 48
00007c1f: (          ): insb byte ptr es:[di], dx ; 656c
00007c21: (          ): insb byte ptr es:[di], dx ; 6c
00007c22: (          ): outsw dx, word ptr ds:[si] ; 6f
00007c23: (          ): sub al, 0x20           ; 2c20
00007c25: (          ): inc bx              ; 43
00007c26: (          ): push 0x6e65         ; 68656e
00007c29: (          ): and byte ptr ds:[eax+117], bl ; 67205875
00007c2d: (          ): and word ptr ds:[bx+si], ax ; 2100
00007c2f: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c31: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c33: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c35: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c37: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c39: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c3b: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c3d: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c3f: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c41: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c43: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c45: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c47: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c49: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c4b: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c4d: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c4f: (          ): add byte ptr ds:[bx+si], al ; 0000
00007c51: (          ): add byte ptr ds:[bx+si], al ; 0000
00007dd7: (          ): add byte ptr ds:[bx+si], al ; 0000
00007dd9: (          ): add byte ptr ds:[bx+si], al ; 0000
00007ddb: (          ): add byte ptr ds:[bx+si], al ; 0000
00007ddd: (          ): add byte ptr ds:[bx+si], al ; 0000
00007ddf: (          ): add byte ptr ds:[bx+si], al ; 0000
00007de1: (          ): add byte ptr ds:[bx+si], al ; 0000
00007de3: (          ): add byte ptr ds:[bx+si], al ; 0000
00007de5: (          ): add byte ptr ds:[bx+si], al ; 0000
00007de7: (          ): add byte ptr ds:[bx+si], al ; 0000
00007de9: (          ): add byte ptr ds:[bx+si], al ; 0000
00007deb: (          ): add byte ptr ds:[bx+si], al ; 0000
00007ded: (          ): add byte ptr ds:[bx+si], al ; 0000
00007def: (          ): add byte ptr ds:[bx+si], al ; 0000
00007df1: (          ): add byte ptr ds:[bx+si], al ; 0000
00007df3: (          ): add byte ptr ds:[bx+si], al ; 0000
00007df5: (          ): add byte ptr ds:[bx+si], al ; 0000
00007df7: (          ): add byte ptr ds:[bx+si], al ; 0000
00007df9: (          ): add byte ptr ds:[bx+si], al ; 0000
00007dfb: (          ): add byte ptr ds:[bx+si], al ; 0000
00007dfd: (          ): add byte ptr ds:[di-86], dl ; 0055aa
<bochs:21>

```

4. 为什么要使用 `jmp $`, 如何改造程序让输出过程执行100次

`jmp $` 是一个无限循环。汇编语言用 `$` 表示地址计数器的当前值，`jmp $` 意思就是跳转到当前的指令地址处执行，即一个死循环。它的作用是使 `hello,os world!` 字符串能持续显示在屏幕上，同时保持系统处于停止状态，防止执行程序之外的代码而发生错误。

改造程序让输出执行100次：

可以通过在汇编代码中增加一个计数器，将字符串输出的操作放入循环中。下面是修改后的代码：

```

mov cx, 100          ; 设置循环次数为100
begin:
    call DispStr    ; 显示字符串
    loop begin      ; 每次减1，直到cx为0
    jmp $
```

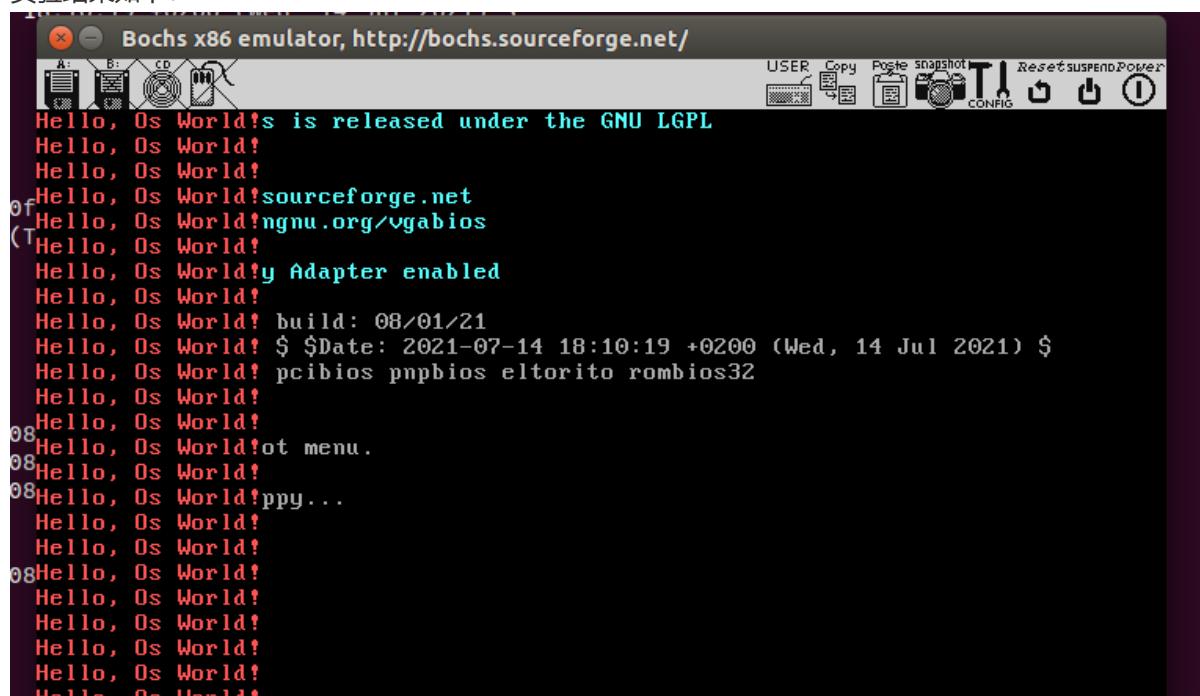
注意到，在DispStr代码段内同样使用了cx寄存器，需要在DispStr开始使用 push cx 保护寄存器，最后使用 pop cx 恢复。

为了能观察到程序输出100次字符串，可以在每次循环时让dh寄存器加一，使得每次在不同的行号输出，完整代码如下：

```

boot.asm x
    org 07c00h           ; where the code will be running
    mov ax, cs
    mov ds, ax
    mov es, ax
    mov cx, 100
    mov dh, 0
begin: call DispStr        ; let's display a string
    loop begin          ; and loop forever
    jmp $
DispStr:
    push cx             ; protect cx register
    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
    add dh, 1
    pop cx
    ret
BootMessage: db "Hello, Os World!"
times 510-($-$) db 0      ; fill zeros to make it exactly 512 bytes
dw 0xaa55                ; boot record signature
```

实验结果如下：



5. 为什么要对段寄存器进行赋值

在x86架构中，段寄存器（如 `CS`、`DS`、`ES`）用于访问不同的内存段。通过使用不同的段寄存器，系统可以将代码、数据和栈等部分分开，避免程序结构混乱和数据相互干扰。引导程序通常在实模式下运行，此时内存地址由段寄存器和偏移量共同决定。

- **段寄存器赋值的原因：**为了确保程序能够正确访问数据，必须将数据段 `DS` 和额外段 `ES` 指向正确的位。例如，在调用 `int 10h` 时，参数 `ES:BP` 指向的是需要显示的字符串数据的起始位置，若 `ES` 未被正确设置，CPU 将无法正确读取字符串数据进行显示。通过给段寄存器赋值，程序能够正确确定位内存中的特定区域。
- 在该程序中，`Hello,os world!` 字符串与代码段位于同一区域，只需将 `es`、`ds` 寄存器与 `cs` 指向同一位置即可。注意段寄存器之间不能直接相互赋值，需要 `ax` 寄存器作为中间寄存器。

6. 如何在该程序中调用系统中断

在汇编程序中，调用 BIOS 系统中断可以通过 `int 中断号` 指令实现。通常，`int 10h` 是用于屏幕显示的中断，常用于输出字符或字符串。

该程序中调用中断的代码：

```
mov ah, 0x13      ; 设置中断号为 0x13 (输出字符串)
mov al, 0x01      ; 显示字符串
mov bx, 0x000c    ; 设置颜色为红色/黑色
mov cx, 16        ; 字符串长度
mov bp, BootMessage ; ES:BP = 字符串地址
mov dl, 0         ; 设置显示位置
int 0x10          ; 调用BIOS中断显示字符串
```

要调用系统中断，需要先确定使用的中断号（这里为 `10h`），并指定 `AH` 寄存器的值，表明需要使用的服务类型。在本例中，设置 `AH` 为 `0x13`，功能为显示字符串。另外，需要设置指定寄存器的值，作为中断程序调用的参数。操作系统通过这些信息来调用中断。

`int 10h` 的参数如下：

`ES:BP = 串地址`

`CX = 串长度`

`DH, DL = 起始行列`

`BH = 页号`

`AL = 0, BL = 属性`

`串: Char, char,, char`

`AL = 1, BL = 属性`

`串: Char, char,, char`

`AL = 2`

`串: Char, attr,, char, attr`

`AL = 3`

`串: Char, attr,, char, attr`

通过 `int 10h`，程序可以向屏幕输出指定的内容。

三、实验过程分析

(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)

问题记录与故障分析

- **问题1：**模拟系统启动时出现 `PANIC`，`module` 缺失
 - **原因分析：**缺失依赖库
 - **解决方法：**安装缺少的库

```
sudo apt-get install bochs-x  
sudo apt-get install bochs-sdl
```

The screenshot shows a terminal window titled "bin : bochs-bin — Konsole". The terminal output is as follows:

```
000000000000i[ ] lt_dlhandle is 0x39f0c80  
000000000000i[PLGIN] loaded plugin libbx_gameport.so  
000000000000i[ ] lt_dlhandle is 0x39f1310  
000000000000i[PLGIN] loaded plugin libbx_iodebug.so  
000000000000i[ ] reading configuration from bochssrc.bxrc  
000000000000i[ ] lt_dlhandle is (nil)  
000000000000p[ ] >>PANIC<< dlopen failed for module 'x': file not found  
=====  
Event type: PANIC  
Device: [ ]  
Message: dlopen failed for module 'x': file not found  
  
A PANIC has occurred. Do you want to:  
cont - continue execution  
alwayscont - continue execution, and don't ask again.  
This affects only PANIC events from device [ ]  
die - stop execution now  
abort - dump core  
debug - continue and return to bochs debugger  
Choose one of the actions above: [die] ^C  
rimeheart@Rimeheart : ~/oslab/lab0/bin $ sudo apt-get install bochs-x  
正在读取软件包列表... 完成  
正在分析软件包的依赖关系树  
A PANIC has occurred. Do you want to:  
cont - continue execution  
alwayscont - continue execution, and don't ask again.  
This affects only PANIC events from device [ ]  
die - stop execution now  
abort - dump core  
debug - continue and return to bochs debugger  
Choose one of the actions above: [die] ^C
```

- **问题2：**执行 `sudo apt-get update` 报错
 - **原因分析：**镜像源和版本不对应，不能随便用一个写入sources.list
 - **解决方法：**执行 `lsb_release -a` 查看对应版本代号（这里是 `trusty`），下载相应版本的 sources.list，覆盖 `/etc/apt/sources.list` 即可

```
ubuntu-32@ubuntu:~/Desktop/bochs-2.7/osfs01-master$ lsb_release -a  
No LSB modules are available.  
Distributor ID: Ubuntu  
Description:    Ubuntu 14.04.6 LTS  
Release:        14.04  
Codename:       trusty  
ubuntu-32@ubuntu:~/Desktop/bochs-2.7/osfs01-master$
```

- **问题3：**执行 `bochs` 报错，无法直接运行 `bochs`，显示尚未安装 `bochs`，但可以通过 `./bochs` 来运行 `bochs`。

- **原因分析**: 在做实验的过程中下载bochs源码包是直接在虚拟机里下载的，在下载路径里解压、安装之后，bochs放在了Download文件夹中，而并未放到系统变量中，导致bochs未能直接运行，而在对应的文件夹中运行 `./bochs` 就可以运行程序，是因为解压在当前目录，解压出来的程序也就在当前目录。

```
yyy@yyy-virtual-machine ~/0/bochs-2.7> bochs
程序“bochs”尚未安装。您可以使用以下命令安装：
sudo apt install bochs
yyy@yyy-virtual-machine ~/0/bochs-2.7> ./bochs
=====
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 '/home/yyy/Downloads/share
/bchs'
00000000000i[      ] reading configuration from .bochsrc
00000000000e[      ] .bochsrc:197: wrong value for parameter 'model'
00000000000p[      ] >>PANIC<< .bochsrc:197: cpu directive malformed.
00000000000e[SIM ] notify called, but no bxevent_callback function is registered
00000000000e[SIM ] notify called, but no bxevent_callback function is registered
=====
Bochs is exiting with the following message:
[      ] .bochsrc:197: cpu directive malformed.
=====
00000000000i[CPU0 ] CPU is in real mode (active)
00000000000i[CPU0 ] CS.mode = 16 bit
00000000000i[CPU0 ] SS.mode = 16 bit
00000000000i[CPU0 ] EFER = 0x00000000
00000000000i[CPU0 ] | EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000
00000000000i[CPU0 ] | ESP=00000000 EBP=00000000 ESI=00000000 EDI=00000000
00000000000i[CPU0 ] | IOPL=0 id vip vif ac vm rf nt of df if tf sf ZF af PF cf
00000000000i[CPU0 ] | SEG sltr(index|ti|rpl) base limit G D
00000000000i[CPU0 ] | CS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | DS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | SS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | ES:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | FS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | GS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | EIP=00000000 (00000000)
00000000000i[CPU0 ] | CR0=0x00000000 CR2=0x00000000
00000000000i[CPU0 ] | CR3=0x00000000 CR4=0x00000000
fish: './bochs' terminated by signal SIGSEGV (Address boundary error)
```

- **解决方法**: 将bochs和bximage程序放到系统变量中，重新运行bochs程序

```
yyy@yyy-virtual-machine ~/0/bochs-2.7> sudo cp -r bochs /usr/local/bin/
[sudo] yyy 的密码:
yyy@yyy-virtual-machine ~/0/bochs-2.7> sudo cp -r bximage /usr/local/bin/
yyy@yyy-virtual-machine ~/0/bochs-2.7> bochs
=====
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 '/home/yyy/Downloads/share
/bchs'
00000000000i[      ] reading configuration from .bochsrc
00000000000e[      ] .bochsrc:197: wrong value for parameter 'model'
00000000000p[      ] >>PANIC<< .bochsrc:197: cpu directive malformed.
00000000000e[SIM ] notify called, but no bxevent_callback function is registered
00000000000e[SIM ] notify called, but no bxevent_callback function is registered
=====
Bochs is exiting with the following message:
[      ] .bochsrc:197: cpu directive malformed.
=====
00000000000i[CPU0 ] CPU is in real mode (active)
00000000000i[CPU0 ] CS.mode = 16 bit
00000000000i[CPU0 ] SS.mode = 16 bit
00000000000i[CPU0 ] EFER = 0x00000000
00000000000i[CPU0 ] | EAX=00000000 EBX=00000000 ECX=00000000 EDX=00000000
00000000000i[CPU0 ] | ESP=00000000 EBP=00000000 ESI=00000000 EDI=00000000
00000000000i[CPU0 ] | IOPL=0 id vip vif ac vm rf nt of df if tf sf ZF af PF cf
00000000000i[CPU0 ] | SEG sltr(index|ti|rpl) base limit G D
00000000000i[CPU0 ] | CS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | DS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | SS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | ES:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | FS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | GS:0000( 0000| 0| 0) 00000000 00000000 0 0
00000000000i[CPU0 ] | EIP=00000000 (00000000)
00000000000i[CPU0 ] | CR0=0x00000000 CR2=0x00000000
00000000000i[CPU0 ] | CR3=0x00000000 CR4=0x00000000
fish: 'bochs' terminated by signal SIGSEGV (Address boundary error)
```

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)

实验结果分析

- **Bochs虚拟机成功搭建**

通过配置 `bochsrc` 文件、安装开发环境和相关依赖库，成功在虚拟机中搭建了 Bochs 环境。在虚拟机中运行 Bochs 并模拟x86架构的启动过程，观察并调试引导扇区程序，验证了实验目标中的环境搭建和调试工具的熟悉度。

- **引导程序成功显示字符串**

编写的 `boot.asm` 程序成功显示了 "Hello, OS world!" 字符串。在进一步的实验中，修改程序输出为个人名字的字符串，并成功通过 `int 0x10` BIOS中断实现屏幕输出。字符串显示效果正确，调试过程顺利完成。

- **汇编语言的调试**

在 Bochs 调试器中，通过设置断点、单步执行和反汇编操作，成功调试了引导程序。观察到程序在内存中的执行流程，理解了引导扇区的作用以及 BIOS 加载引导程序的过程。

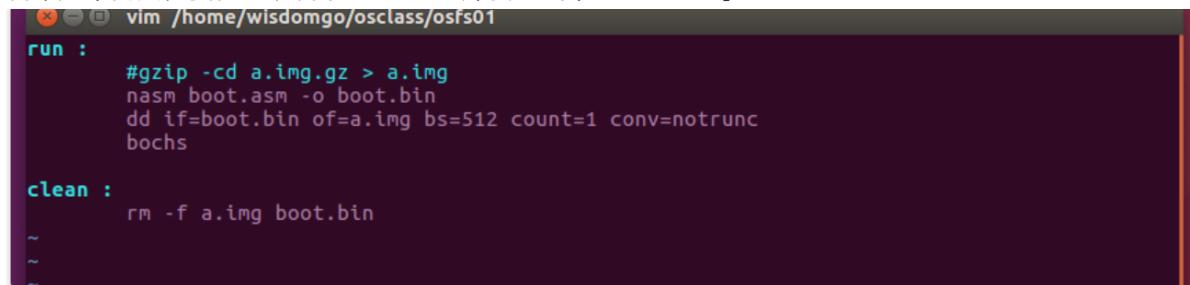
改进建议

- 改进1：书上的 bochs 版本有一些落后，如果想要使用新版本的 bochs 做实验的话，需要根据官方文档来调整 BIOS 和显示驱动所在的路径。也许可以把官方的[编译手册](#)放入实验参考资料中，便于后面的同学编译。
- 改进2：在循环中每次调用系统中断之前，修改 DH 寄存器的值，即能实现多行打印的效果。

FQA

? 我一定需要使用bximage创建软驱吗

答案是不需要，我们可以看下clone的仓库代码当中的makefile[



```
vim /home/wisdomgo/osclass/osfs01
run :
    gzip -cd a.img.gz > a.img
    nasm boot.asm -o boot.bin
    dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
    bochs

clean :
    rm -f a.img boot.bin
~
```

不难发现，make的第一行已经将仓库中的压缩包解压为了一个软驱，实际上这一步也不是必要的，因为第三行会将二进制文件写入一个a.img，即使没有a.img也会为我们创建一个。

? apt安装一些包时产生依赖无法解决

apt产生依赖报错一般是源的问题，新手很容易出现源的版本和ubuntu版本不匹配问题。检查步骤如下

命令行输入 `lsb_release -a` 命令，查看codename (以ubuntu14.04为例)

```
No LSB modules are available.  
Distributor ID: Ubuntu  
Description:    Ubuntu 14.04.6 LTS  
Release:        14.04  
Codename:       trusty
```

可以看到对应的code那么为trusty，查看/etc/apt/sources.list中对应的源版本，检查是否一致，如果为xenial说明使用了ubuntu16的apt源。

```
deb http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ trusty main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ trusty-security main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ trusty-updates main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ trusty-proposed main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ trusty-backports main restricted universe multiverse
```

？如何在虚拟机和宿主机之间传递文件

- 共享文件夹
- vmtools
- xshell + xftp连接

？安装bochs源码时迟迟未弹出下载窗口

由于网络的问题，在sourceforge下载bochs源码时确实需要等待一段时间，可以使用wget命令，直接在命令行中输入 wget <https://sourceforge.net/projects/bochs/files/bochs/2.7/bochs-2.7.tar.gz> 进行下载

？如何找到vgaromimage的对应文件位置

使用find命令在根目录下寻找文件"BIOS-bochs-latest"和"vgabios.bin"文件。

```
wisdomgo@ubuntu ~> sudo find -name "BIOS-bochs-latest"  
. ./bochs-2.7/bios/BIOS-bochs-latest  
wisdomgo@ubuntu ~>
```

写入bochssrc文件

五、各人实验贡献与体会

1. 黄东威：

○ 负责任务:

独立完成全部实验，参与撰写实验报告，承担了FAQ任务，在实验安排的基础任务上，主动的修改汇编代码，bochs文件，makafile等进行尝试和探索，同时和组内成员一起讨论了完成实验任务的心得。

○ 个人体会：

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。

2. 王浚杰：

○ 负责任务:

独立完成全部实验，完成实验后交流了实验心得，撰写实验报告的主体部分。

○ 个人体会：

本次实验通过搭建 Bochs 虚拟机环境，学习了操作系统引导的基本原理和 BIOS 的启动过程。通过编写汇编代码，第一次感受到低级程序是如何与硬件交互的。在虚拟机中成功运行并调试了引导扇区程序，并通过调试工具对程序进行断点调试和反汇编操作，直观感受到 CPU 如何逐条执行汇编指令，并能看到每一步的内存和寄存器变化。这让我对程序的执行流程有了更深的理解。

实验过程中也遇到了一些问题，比如最初忘记安装依赖库，导致 Bochs 无法启动，最后通过查找资料得以解决问题。此次实验让我对操作系统底层原理有了更深入的理解。

3. 程序：

- **负责任务：**

独立完成全部实验，撰写报告中实验练习的4、5、6题，第3题的补充部分，补充了部分实验中遇到的问题。

- **个人体会：**

本实验学习了实验的基本环境搭建，通过一小段简单的操作系统代码基本回顾了汇编语言的知识，理解了BIOS的启动过程并制作了一个可启动的软盘，同时学习使用了bochs工具来调试操作系统。通过这次实验，我对ubuntu系统及其环境搭建有了更深的认识，比如ubuntu每一个版本都有一个代号，国内源必须要跟这个代号对应，否则会出现各种各样的问题等等。

4. 周业营：

- **负责任务：**

独立完成全部实验，交流实验心得，提出自己在实验过程中遇到的问题及对应的解决方案。

- **个人体会：**

在做这次实验的过程中，第一次自己实现了一个虚拟软驱的创建、配置和运行。学习和掌握了使用bochs环境来运行、调试引导程序。实验中也遇到了许多问题，从bochs的安装到bochsrc文件和boot文件的修改，在不断的试错和改进的过程中不断增进自己对每一部分代码的认知，了解每部分代码实现的功能并观察修改代码之后的运行变化。在做实验的过程中，实现融会贯通操作系统、汇编语言和Linux系统的知识，在实操中进一步掌握理论知识，构建自己的知识网络，提升自己对计算机的认识。
