

# OS LAB 1 系统引导

## 个人信息

姓名：郑凯琳

学号：205220025

邮箱：[205220025@smail.nju.edu.cn](mailto:205220025@smail.nju.edu.cn)

## 1. 实验目的

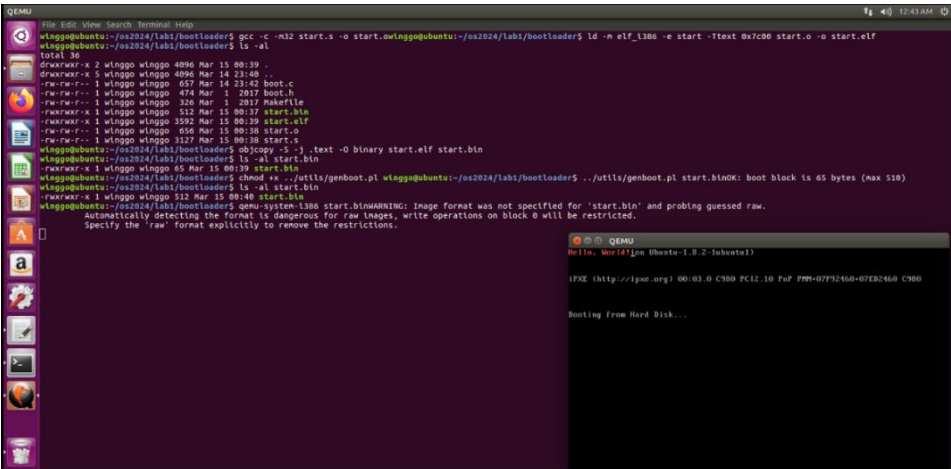
- (1) 学习在 Linux 环境下编写、调试程序，掌握 Shell、Vim、GCC、Binutils、Make、QEMU、GDB 的使用。
- (2) 学习 AT&T 汇编程序的特点
- (3) 理解系统引导的含义，理解系统引导的启动。

## 2. 实验进度

我完成了所有内容，成功地在实模式、保护模式及保护模式加载磁盘的条件下运行 Hello World 程序。

## 3. 实验结果

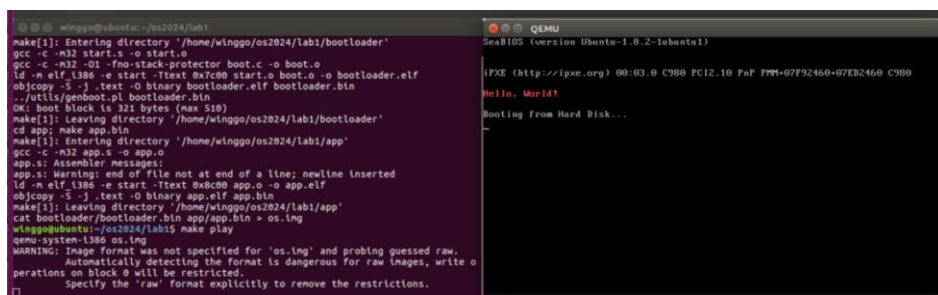
- (1) 在实模式下实现一个 Hello World 程序
- 在实模式下在终端中打印 Hello, World!



创建一个主引导扇区，然后用 qemu 启动它，并在屏幕上输出 Hello, World!

(2) 在保护模式下实现一个 Hello World 程序

从实模式切换至保护模式，并在保护模式下在终端中打印 Hello, World!



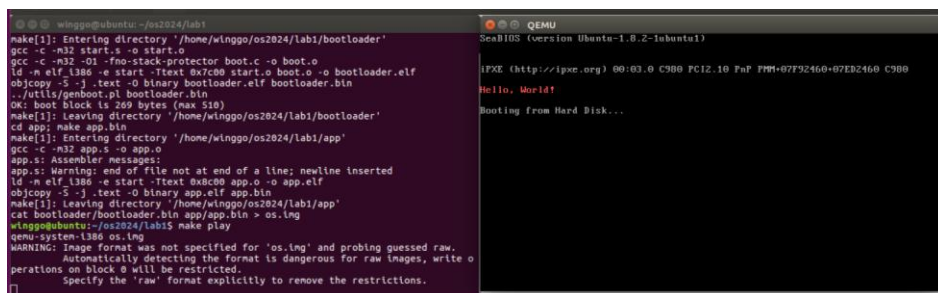
```
make[1]: Entering directory '/home/winggo/os2024/lab1/bootloader'
gcc -c -m32 start.s -o start.o
gcc -c -m32 -D_FPU_STACK_PROTECTOR boot.c -o boot.o
ld -m elf_i386 -e start -Ttext 0x7C00 start.o boot.o -o bootloader.elf
objcopy -S -j .text -O binary bootloader.elf bootloader.bin
./utils/genboot.pl bootloader.bin
OK: boot block is 512 bytes (max 510)
make[1]: Leaving directory '/home/winggo/os2024/lab1/bootloader'
cd app; make app.bin
make[1]: Entering directory '/home/winggo/os2024/lab1/app'
gcc -c -m32 app.s -o app.o
app.s: Assembler messages:
app.s: Warning: end of file not at end of a line; newline inserted
ld -m elf_i386 -e start -Ttext 0x8000 app.o -o app.elf
objcopy -S -j .text -O binary app.elf app.bin
make[1]: Leaving directory '/home/winggo/os2024/lab1/app'
cat bootloader/bootloader.bin app/app.bin > os.img
winggo@ubuntu:~/os2024/lab1$ make play
qemu-system-i386 os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
[
[Seabios (version Ubuntu-1.8.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03:0 C900 FC12:10 FaP FMM+07F92460+07ED2460 C900
Hello, World!
Booting from Hard Disk...
```

在系统启动时首先进入 8086 的实模式，但是由于实模式下的安全性和其分段机制的问题，在 80386 中引入了保护模式。

模拟了 8086 的实模式到 80386 的保护模式的切换。

(3) 在保护模式下加载磁盘中的 Hello World 程序运行

从实模式切换至保护模式，在保护模式下读取磁盘 1 号扇区中的 Hello World 程序至内存中的相应位置，跳转执行该 Hello World 程序，并在终端中打印 Hello, World!



```
make[1]: Entering directory '/home/winggo/os2024/lab1/bootloader'
gcc -c -m32 start.s -o start.o
gcc -c -m32 -D_FPU_STACK_PROTECTOR boot.c -o boot.o
ld -m elf_i386 -e start -Ttext 0x7C00 start.o boot.o -o bootloader.elf
objcopy -S -j .text -O binary bootloader.elf bootloader.bin
./utils/genboot.pl bootloader.bin
OK: boot block is 269 bytes (max 510)
make[1]: Leaving directory '/home/winggo/os2024/lab1/bootloader'
cd app; make app.bin
make[1]: Entering directory '/home/winggo/os2024/lab1/app'
gcc -c -m32 app.s -o app.o
app.s: Assembler messages:
app.s: Warning: end of file not at end of a line; newline inserted
ld -m elf_i386 -e start -Ttext 0x8000 app.o -o app.elf
objcopy -S -j .text -O binary app.elf app.bin
make[1]: Leaving directory '/home/winggo/os2024/lab1/app'
cat bootloader/bootloader.bin app/app.bin > os.img
winggo@ubuntu:~/os2024/lab1$ make play
qemu-system-i386 os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write
operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
[
[Seabios (version Ubuntu-1.8.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03:0 C900 FC12:10 FaP FMM+07F92460+07ED2460 C900
Hello, World!
Booting from Hard Disk...
```

## 4. 实验修改的代码位置

(1) 实模式下实现一个 Hello World 程序

start.s

```

.code16
.global start
start:
    movw %cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    movw $0x7d00, %ax
    movw %ax, %sp # setting stack pointer to 0x7d00
    # TODO:通过中断输出Hello World
    pushw $13 # pushing the size to print into stack
    pushw $message # pushing the address of message into stack
    callw displayStr # calling the display function

loop:
    jmp loop

message:
    .string "Hello, World!\n\0"

displayStr:
    pushw %bp
    movw 4(%esp), %ax
    movw %ax, %bp
    movw 6(%esp), %cx
    movw $0x1301, %ax
    movw $0x000c, %bx
    movw $0x0000, %dx
    int $0x10
    popw %bp
    ret

```

通过中断来实现输出 "Hello, World!"。通过调用 BIOS 中断 0x10，并设置 AH 寄存器的值为 0x13，以及设置其他必要的寄存器值，可以实现在屏幕上显示文本。

displayStr 函数负责将字符串从内存中读取并输出到屏幕上。

## (2) 在保护模式下实现一个 Hello World 程序

start.s

```

.code16
.global start
start:
    movw %cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    # TODO:关闭中断
    cli

    # 启动A20总线
    inb $0x92, %al
    orb $0x02, %al
    outb %al, $0x92

    # 加载GDTR
    data32 addr32 lgdt gdtDesc # loading gdt, data32, addr32

    # TODO:设置cr0的PE位(第0位)为1
    movl %cr0, %eax
    orb $0x01, %al
    movl %eax, %cr0

    # 长跳转切换至保护模式
    data32 ljmp $0x00, $start32 # reload code segment selector and jmp to start32, data32

```

- 1) 使用 cli 关闭中断。
- 2) 启动 A20 总线。

如果 A20 地址线未开启，地址的第 20 位将永远为 0，导致地址空间不连续。

- 3) 加载 GDTR 寄存器。
- 4) 设置 CR0 的 PE 位（第 0 位）为 1。启动保护模式。
- 5) 长跳切换至保护模式。

CS 不可直接修改。

```
code32
start32:
    movw $0x10, %ax # setting data segment selector
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %fs
    movw %ax, %gs
    movw $0x18, %ax # setting graphics data segment selector
    movw %ax, %gs

    movl $0x8000, %eax # setting esp
    movl %eax, %esp
    # TODO:输出Hello World
    pushl $13 # pushing the size to print into stack
    pushl $message # pushing the address of message into stack
    calll displayStr # calling the display function

loop32:
    jmp loop32

message:
    .string "Hello, World!\n0"

displayStr:
    movl 4(%esp), %ebx
    movl 8(%esp), %ecx
    movl $(80*5+9)*2, %edi
    movb $0xd, %ah

nextchar:
    movb (%ebx), %al
    movw %ax, %gs:*(%edi)
    addl $2, %edi
    incl %ebx
    loopnz nextchar # loopnz decrease ecx by 1
    ret
```

在 32 位模式下，指令从 w 变成 l，表示操作双字数据。

通过循环控制将 `al` 不断地设置为需要打印的 ASCII 字符，然后使用 `movw %ax, %gs:(%edi)` 来写入显存。

```

p2align 2
gdt: # 8 bytes for each table entry, at least 1 entry
    .word limit[15:0],base[15:0]
    # byte base[23:16],0x90(type),0xc0(lim[19:16]),base[31:24]
    # c0第一个表项为空
    .word 0,0
    .byte 0,0,0,0

# TODO: code segment entry
    .word 0xffff,0
    .byte 0,0x9a,0xcf,0

# TODO: data segment entry
    .word 0xffff,0
    .byte 0,0x92,0xcf,0

# TODO: graphics segment entry
    .word 0xffff,0x8000
    .byte 0x0b,0x92,0xcf,0

gdtDesc:
    .word (gdtDesc - gdt - 1)
    .long gdt

```

定义了全局描述符表（Global Descriptor Table, GDT），用于存储段描述符的数据结构、管理内存分段和保护。

- 代码段描述符
- 数据段描述符
- 图形段描述符

(3) 在保护模式下加载磁盘中的 Hello World 程序运行

start.s

```

.code16
.global start
start:

    movw %Cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    # TODO: 关闭中断
    cli

    # 启动A20总线
    inb $0x92, %al
    orb $0x02, %al
    outb %al, $0x92

    # 加载GDT
    data32 addr32 lgdt gdtDesc # loading gdt, data32, addr32

    # TODO : 设置cr0的pe位 (第0位) 为1
    movl %cr0, %eax
    orb $0x01, %al
    movl %eax, %cr0

    # 长路转切换至保护模式
    data32 ljmp $0x08, %start32 # reload code segment selector and ljmp to start32, data32

```

```

.code32
start32:
    movw $0x10, %ax # setting data segment selector
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %fs
    movw %ax, %ss
    movw $0x18, %ax # setting graphics data segment selector
    movw %ax, %gs

    movl $0x8000, %eax # setting esp
    movl %eax, %esp
    jmp bootMain # jump to bootMain in boot.c

.p2align 2
gdt: # 0 bytes for each table entry, at least 1 entry
    # .word limit[15:0],base[15:0]
    # .byte base[23:16],(0x90|(type)),(0xc0|(limit[19:16])),base[31:24]
    # gdt第一个表项为空
    .word 0,0
    .byte 0,0,0,0

    # T000: code segment entry
    .word 0xffff,0
    .byte 0,0x9a,0xcf,0

    # T000: data segment entry
    .word 0xffff,0
    .byte 0,0x92,0xcf,0

    # T000: graphics segment entry
    .word 0xffff,0x8000
    .byte 0x0b,0x92,0xcf,0

gdtDesc:
    .word (gdtDesc - gdt - 1)
    .long gdt

```

同 (2)。

boot.c

```

void bootMain(void) {
    //TODO
    void (*elf)(void);
    elf = (void*)(void)0x8c00;
    readSect((void*)elf, 1); // loading sector 1 to 0x8c00
    elf(); // jumping to the loaded program
}

```

在 boot.c 中调用 readSect(void \*dst, int offset)函数，通过 boot.h 中封装的读写接口，完成磁盘中 1 号扇区中 Hello World 程序的加载，并使用函数指针的方式执行该程序。

注：将 elf 赋值为 0x8c00 是因为在 app 的 Makefile 中可以看到其入口地址就是 0x8c00。