**操作系统实验报告**

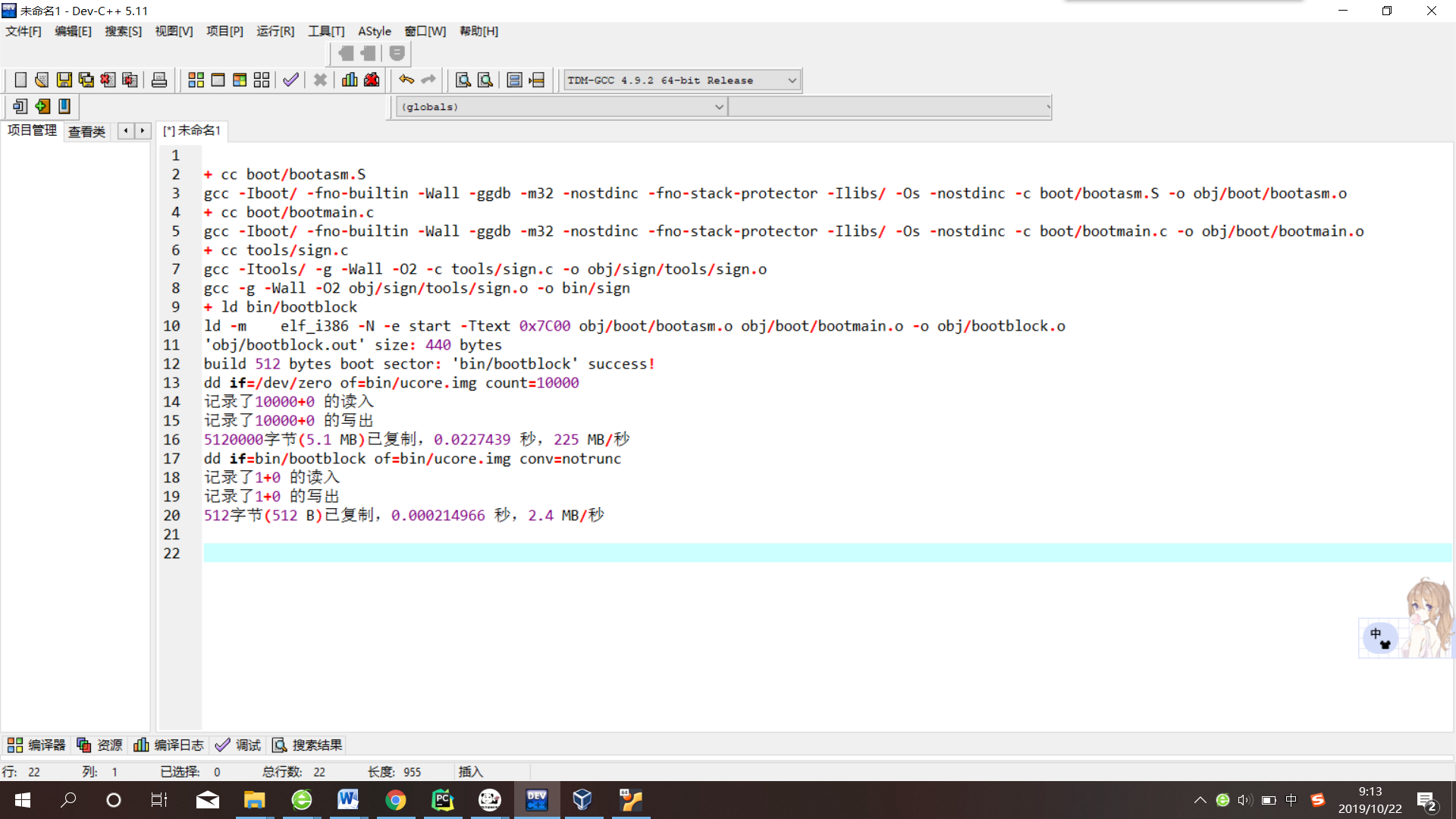
171491221 姜世廷

**练习1，理解通过make生成执行文件的过程。（要求在报告中写 出对下述问题的回答）**

列出本实验各练习中对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了 原理中的基本概念和关键知识点。

1. 操作系统镜像文件ucore.img是如何一步一步生成的？(需要比较详细地解释Makefile中每 一条相关命令和命令参数的含义，以及说明命令导致的结果)

**执行$ make "V="：**



生成ucore.img首先需要生成bootblock，而生成bootblock需要先生成bootmain.o和bootasm.o还有sign，这三个文件又分别由bootmain.c、bootasm.S、sigh.c来生成。

ld -melf\_i386 -N -e start -Ttext 0x7C00 obj/boot/bootasm.o obj/boot/bootmain.o –o obj/bootblock.o这句话用于生成bootblock，elf\_i386表示生成elf头，0x7C00为程序的入口。

'obj/bootblock.out' size: 440 bytes

这句话表示生成的bootblock的文件大小，需要给blootblock填充，，最后两字节设置为了0x55，0xAA

buf[510] = 0x55;

buf[511] = 0xAA;

FILE \*ofp = fopen(argv[2], "wb+");

size = fwrite(buf, 1, 512, ofp);

**练习2：使用qemu执行并调试lab1中的软件。（要求在报告中简要写出练习过程）**

1. 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

(1)修改shiyan1/tools/gdbinit，内容为：

Set architecture i8086

Target remote :1234

(2)在shiyan1目录下执行make debug

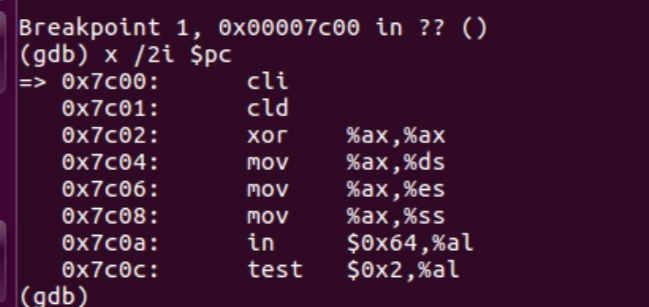
(3)在看到gdb的调试界面后，执行命令si,即可单步跟踪BIOS

(4)在gdb界面下，执行x/2i $pc来查看BIOS的代码

2. 在初始化位置0x7c00设置实地址断点,测试断点正常。

运行make debug可得:

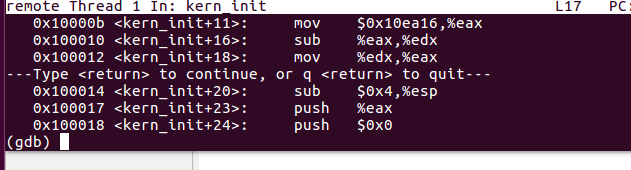
断点正常：



3.从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和bootblock.asm进行比较。

比较结果：相同

4. 自己找一个bootloader或内核中的代码位置，设置断点并进行测试。



**练习3：分析boot loader进入保护模式的过程**

BIOS将通过读取硬盘主引导扇区到内存，并跳转到对应内存中的位置执行boot loader。请分析boot loader是如何完成从实模式进入保护模式的。

需了解：为何开启A20，以及如何开启A20；如何初始化GDT表；如何使能和进入保护模式

* 开启A20：通过将键盘控制器上的A20线置于高电位，使全部32条地址线可用，进而可以访问4G的内存

seta20.1:   # 等待8042键盘控制器不忙

 inb $0x64…

* 初始化GDT表：GDT表和其描述符已经静态储存在引导区中，载入即可

lgdt gdtdesc

* 进入保护模式：通过将cr0寄存器PE位置1便开启了保护模式

movl %cr0,%eax

orl $CR0\_PE\_ON,%eax

movl %eax,%cr0

    从实模式转换到保护模式，用到了全局描述符表和段表，使得虚拟地址和物理地址匹配，保证转换时有效的内存映射不改变；lgdt汇编指令把GDTR描述符表的大小和起始位置存入gdtr寄存器中；将CR0的最后一位设置为1，进入保护模式；指令跳转由代码段跳到protcseg的起始位置。