实验项目 3： 汇编与C混合编程

姓名：欧阳润宇 学号：16337188 邮箱：a59135996@qq.com

院系：数据科学与计算机学院 专业：16级计算机科学与技术 指导教师： 郑培嘉

# 【实验题目】

汇编与C混合编程

# 【实验目的】

学习汇编与C的混合编程，了解FAT12文件系统

# 【实验要求】

# 用汇编写一些函数，在C中调用他们；用C写一些函数，在汇编中调用他们。

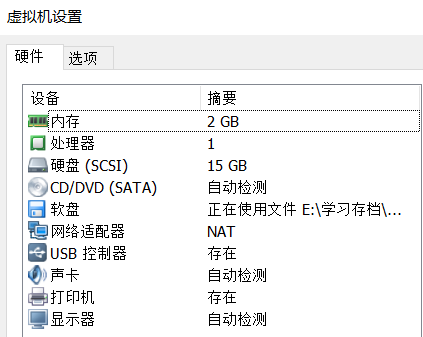
（2）引入文件系统

（3）解读FAT12

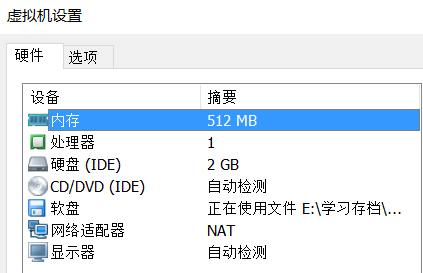
# 【实验方案】

## 硬件及虚拟机配置

Ubuntu:



裸机：



## 软件工具及作用

VMware: 虚拟机

Vscode: 编辑器

WinHex: 16进制编辑器

Nasm: 汇编器

## 程序功能（具体操作详见README）

1. 将“heLlo wOrld!”转化为全大写的“HELLO WORLD”
2. 输入4个数字（数字范围为0~2^31）输出它们的最大值和最小值
3. 清屏
4. 输入测试：

第一次输入是非阻塞且不回显的，但处于一个无限循环之中，只有输入字符才可退出循环，输入的字符会留在缓冲区。

第二次输入是阻塞且回显的，但你无需输入，因为它会读取第一次输入留在缓冲区的字符。

第三次输入是非阻塞但回显的，但处于一个无限循环之中，只有输入字符才可退出循环，输入的的字符回显并留在缓冲区。

第四次输入是阻塞但不回显的，你无需输入，因为它会读取第三次输入留在缓冲区的字符。

1. 加载子程序并运行

第一个子程序的功能是，输入两个数（0~65535），输出它们的和（和的范围也必须是0~65535）

第二个子程序的功能是，输入一个数（0~65535），输出小于等于该数的所有质数

第三个子程序的功能是，在屏幕上显示一个“HELLO”的图形

每个子程序在结束后按Esc都可以回到测试程序

1. 打印所有文件名字

## 程序设计

C和汇编混合编程部分

**void clr\_screen();**

功能：清屏



**int key\_detect(int echo, int block);**

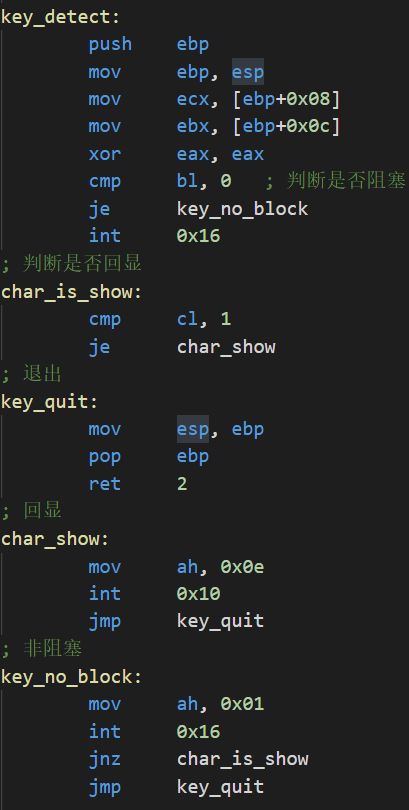
功能：按键检测

参数说明：

echo —— 指定检测到某⼀按键是否在屏幕上显⽰检测到的字符；

block —— 指定调⽤该函数时是否阻塞，即是否等待任意按键再继续执⾏；

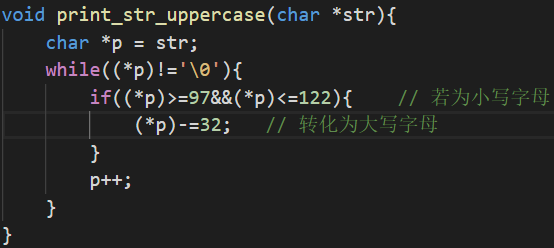
返回值：检测得到的按键值，未检测到返回0



**void print\_str\_uppercase(char \*str)**

功能：

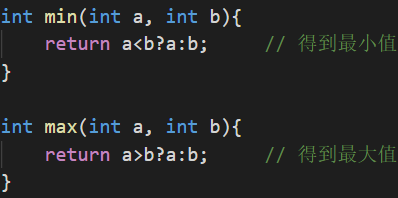
若str不为空指针（str != 0），则将地址str所指向的字符串转为⼤写。

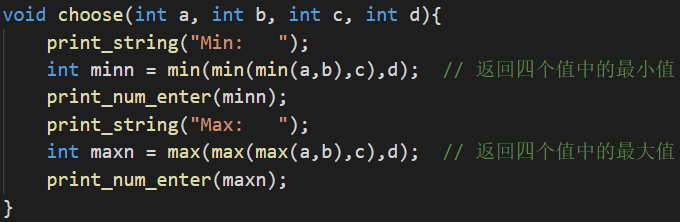


**void choose(int a, int b, int c, int d)**

功能：

打印输出a, b, c, d中的最⼤值和最⼩值





**void read\_floppy(int segStart, int segNum, void\* addr);**

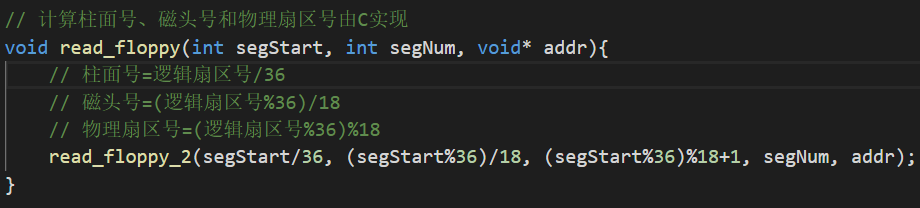
功能：读软盘功能

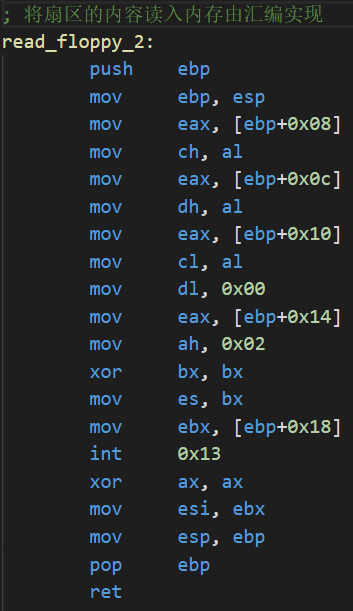
参数说明：

segStart——从软盘的segStart扇区开始将软盘读⼊内存；

segNum——读取软盘的扇区数

addr——写入内存的首地址





（utils.asm中定义的函数在main.c中测试，main.c中定义的函数在main.c和entry.asm中测试，测试方法详见代码文件）

文件系统部分（包括FAT12的解读）：

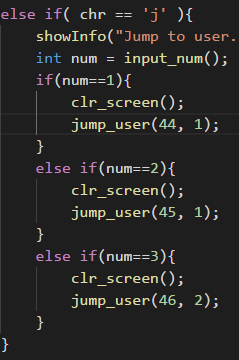
未解读FAT12格式时，先将软盘加载到Ubuntu，再将三个子程序和生成的kernel.bin放到带文件系统的软盘映像中，然后卸载软盘。将得到的软盘映像用16进制编辑器查看，得到每个子程序的起始扇区号、名字和扇区数。

**void jump\_user(int start, int num);**

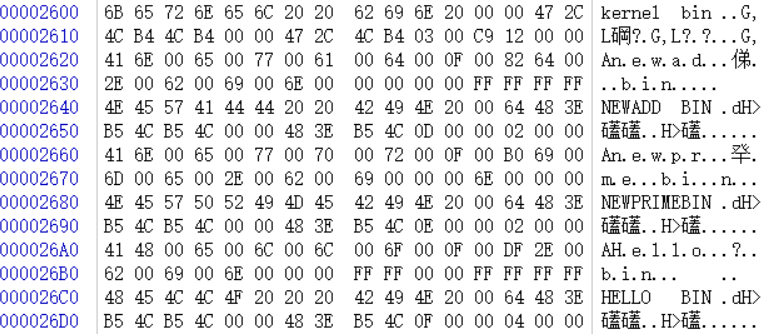
功能：将起始扇区号为start，num个的扇区的内容加载到内存8100h处运行

实现方法类似于前面的read\_floppy。

在未解读FAT12格式时，加载子程序的方法为：



解读FAT12的文件目录表（以下分析只适用于Ubuntu）：

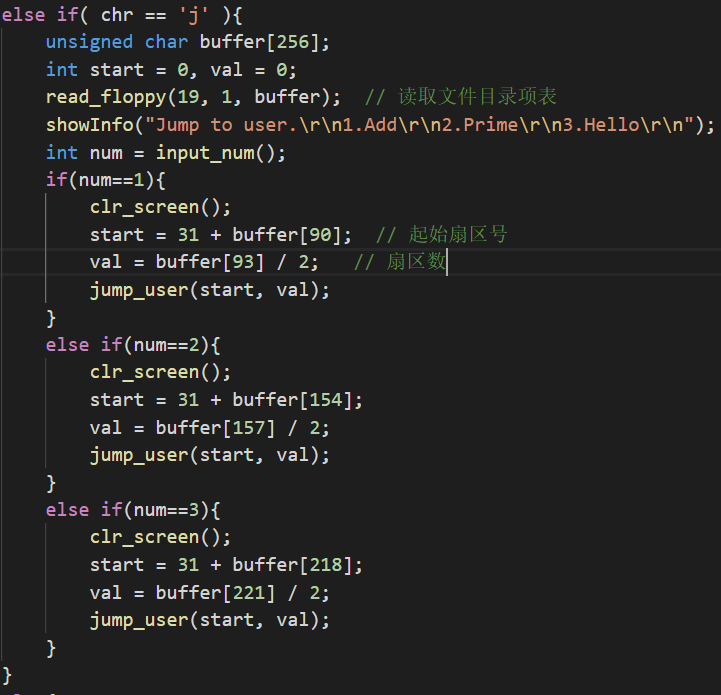


根据文件目录项表可知，除kernel.bin外，每64bytes储存一个文件的信息，但只有后32bytes是有效的。在后32bytes里，前8bytes是文件名（不足8bytes的用0x20补充），第9~11bytes是扩展名。

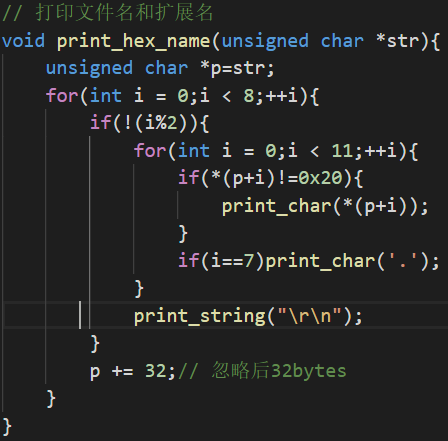
第27~28bytes表示起始扇区（注意基础扇区为31扇区，而且这里27~28bytes的顺序发生了颠倒，比如NEWADD.BIN中对应位置的0x0D00实际上储存的信息是0x000D，表示NEWADD.BIN的起始扇区是第31+0x0D扇区）。

第29~32bytes表示文件大小，单位为bytes（注意这里的顺序也发生了颠倒，例如kernel.bin中对应的0xC9120000实际上存储的信息是0x000012C9，表明kernel.bin的文件大小是4809bytes）。根据文件大小（即第30bytes）可以推算出文件占用的扇区数（例如NEWADD.BIN的文件大小实际上为0x00000200，即512bytes，由于一个扇区也是512bytes，所以可以知道NEWADD.BIN占用的扇区数为1）

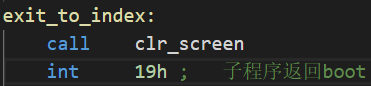
解读了FAT12之后，先解读相应子程序的文件目录项，得到子程序的起始扇区和扇区数，然后再装载子程序并运行。



同样，可以通过解读文件目录项表得到子程序的文件名和扩展名。



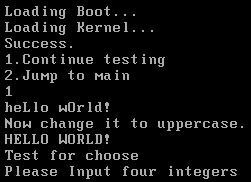
子程序返回main的方法：



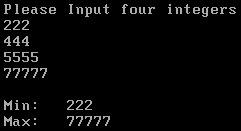
# 【实验过程】

## 功能展示（运行前请看README）

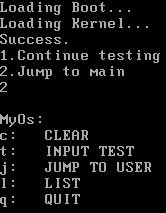
开始界面以及测试小写转大写功能：



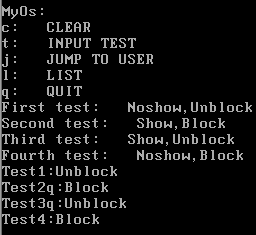
测试四个数中选最大值和最小值的功能：



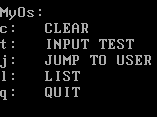
跳转到Main函数的测试部分：



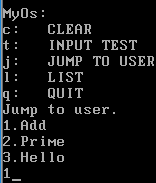
输入测试：



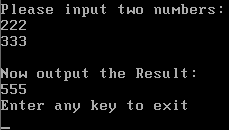
清屏：



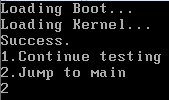
装载子程序并运行：



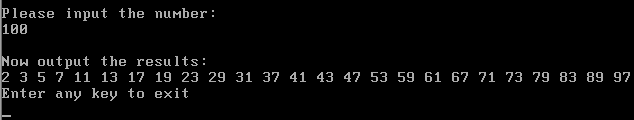
子程序1：



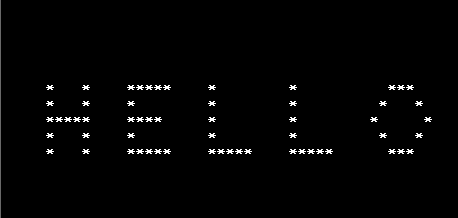
子程序1返回：



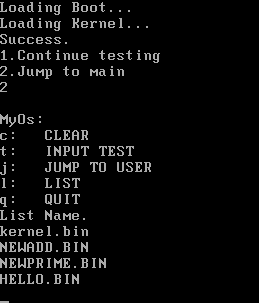
子程序2：



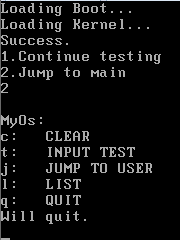
子程序3：



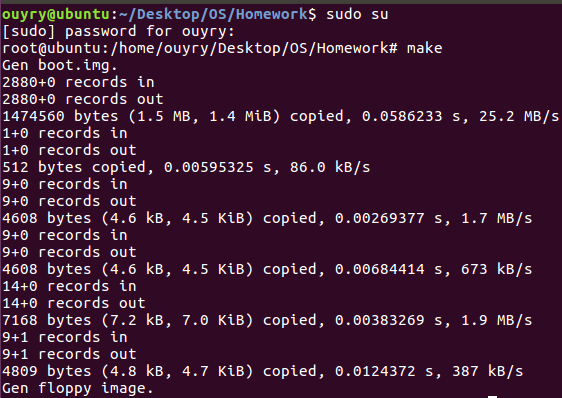
打印文件名：



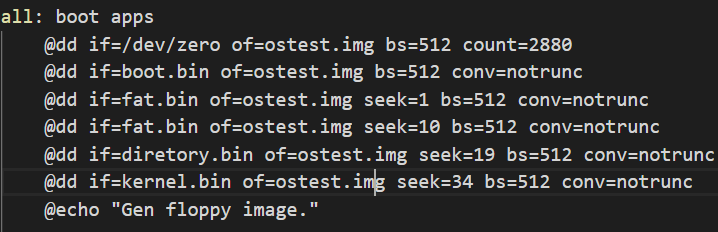
退出：

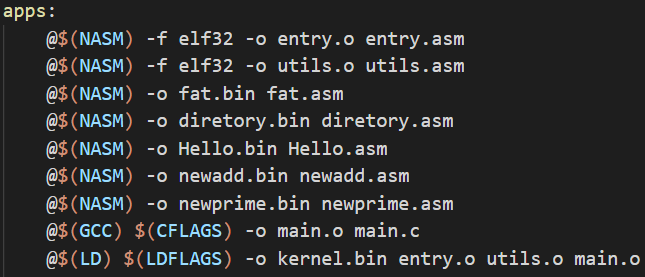


## 编译过程（编译在Ubuntu16.04.2\_i384虚拟机上进行，编译前请看README）



Makefile修改部分：



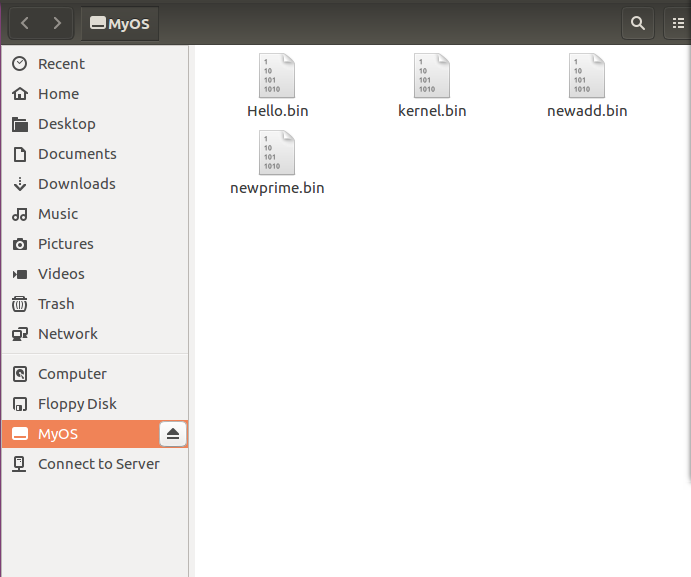


## 映像写入

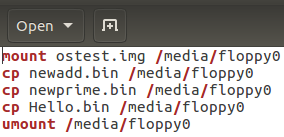


此时装载硬盘，可以看到：





test.sh内容（test.sh必须在root条件下运行）：



（功能是将生成的ostest.img加载到软盘中，并将生成的三个子程序的.bin文件复制到软盘中，再卸载软盘）

## 虚拟机设置

（见前文硬件及虚拟机配置部分）

# 【实验总结】

这次实验分成必做和选做两个部分。

在必做部分，遇到了两个问题。第一个问题是不懂key\_detect的第一个参数是什么意思，不了解阻塞和非阻塞的区别，后来看了TA的补充文档才懂。第二个问题是在写读软盘功能时，一开始选择直接用汇编来完成，但由于汇编做除法有诸多不方便，导致逻辑写得非常乱，花费了大量时间debug，最后决定将逻辑扇区和物理扇区的转换在C中完成，汇编只负责传参和加载扇区到内存，才终于将这个函数完成。

在选做部分，遇到的问题更多。在引入文件系统时，一开始并没有弄清楚TA的意思，用一个没有文件系统的img文件加载到软盘，无论如何拖动文件都无法将文件放入软盘，而且将程序直接拖进软盘的步骤总是出错，在这里花费了大量时间。后来修改了下makefile并学习了linux的mount命令，才解决了这一部分的问题。

后来又出现了跳转的问题，即如何从boot跳到kernel，如何main跳转到子程序，并从子程序跳回main。一开始是弄错了x86冒号的寻址方式，导致子程序跳回main的部分老是失败，后来问了同学才解决了这一问题。再后来又出现了加载内存的问题，即不知道如何分配内存给各个程序，导致main可以跳转到子程序，但子程序的功能却不能正常进行，后来更改了加载的内存地址，又导致子程序的功能可以正常运行，但无法跳回main的错误。这两个情况最终也没能解决。而是固定子程序加载到8100h,并用重启中断int 19h，让它直接跳回到boot，才解决了子程序跳转回kernel的功能。

最后在解析FAT12时，为了让kernel.bin一开始就能在软盘中显示出来，在解析FAT表上花费了大量时间，也没能弄清FAT表是如何生成的，最终只能将Ubuntu生成的FAT表复制粘贴，在编译时将它偏移到相应的扇区，才解决了这一问题。

总而言之，这次实验的必做部分较为简单，但我在选做部分花费了整整一个周末的时间，这给了我一个很大的教训——自己太菜了就不要逞强做什么选做题。

# 【参考文献】

所有代码及其生成文件见附件中的code.zip

<https://blog.csdn.net/b2b160/article/details/4394907>

<http://blog.sina.com.cn/s/blog_6fb22e5b01010x6q.html>

<http://blog.sina.com.cn/s/blog_541e0ada0100ltvg.html>

<http://www.disc.ua.es/~gil/FAT12Description.pdf>

https://www.cnblogs.com/heisenberg-/p/6582124.html