浙江大学 实验报告

课程名称：嵌入式系统　　指导老师：蔡铭　　学生姓名：李磊

实验名称：Lab 5：Linux 系统调用　实验类型：操作实践　　学生学号：3110102782

一、实验目的和要求

实验目的：

1. 学习 Linux 内核的配置和编译；
2. 深入理解 Linux 系统调用；
3. 理解 ARM 和 x86 的 CPU 模式（系统模式、用户模式等）的不同。

实验要求：

使用交叉编译工具或本机编译工具，通过 Linux 内核源码进行实验:

1. 寻找、下载 pcDuino 的 Linux 内核源码；
2. 在内核中加入新的系统调用，具体功能没有要求，能输出调试信息即可;
3. 修改内核代码配置，编译内核；
4. 将编译好的内核装载到 pcDuino 启动；
5. 编写 C 代码，用两种方法做系统调用，测试：
   a) 嵌入汇编代码，用 r0 传参数；
   b) 用 syscall()函数。

二、实验内容和原理

使用交叉编译工具或本机编译工具，通过 Linux 内核源码进行实验:

1. 寻找、下载 pcDuino 的 Linux 内核源码；
2. 在内核中加入新的系统调用，具体功能没有要求，能输出调试信息即可;
3. 修改内核代码配置，编译内核；
4. 将编译好的内核装载到 pcDuino 启动；
5. 编写 C 代码，用两种方法做系统调用，测试：
   c) 嵌入汇编代码，用 r0 传参数；
   d) 用 syscall()函数。

三、主要仪器设备

1．Raspberry Pi 主板一块；
2．5V/1A 电源一个；
3．microUSB 线一根；
4．USB-TTL 串口线一根（PL2303 芯片）；
5．PC（Windows/Mac OS/Linux）一台；
6．以太网线一根；
7．路由器一台；
8．HDMI 显示器；
9．HDMI 线；
10．USB 键盘/鼠标。

四、操作方法和实验步骤

1. 寻找、下载 pcDuino 的 Linux 内核源码

```
$ git clone https://github.com/raspberrypi/linux.git
$ git clone https://github.com/raspberrypi/firmware.git
```

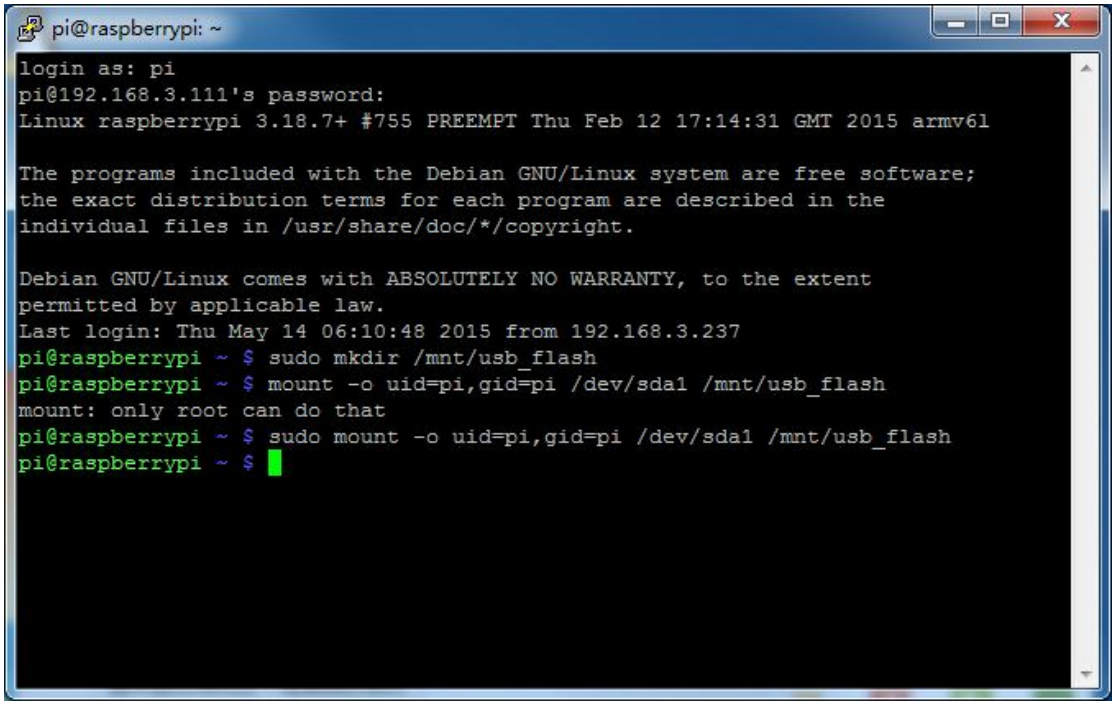
然而。。。。。。在树莓派上连续下载了 3 次都不成功，下载到一半提示下载错误，真是哭死了。

然后，我开始尝试各种各样的方法，现在 windows 下在 github 网站下载 zip 文件，下载完发现根本没有源码，然后又安装了 github 软件，还是不成功，后来我想了一个方法，用虚拟机里的 linux 操作系统去 git clone，git clone 了一天一夜，终于 git clone 成功。

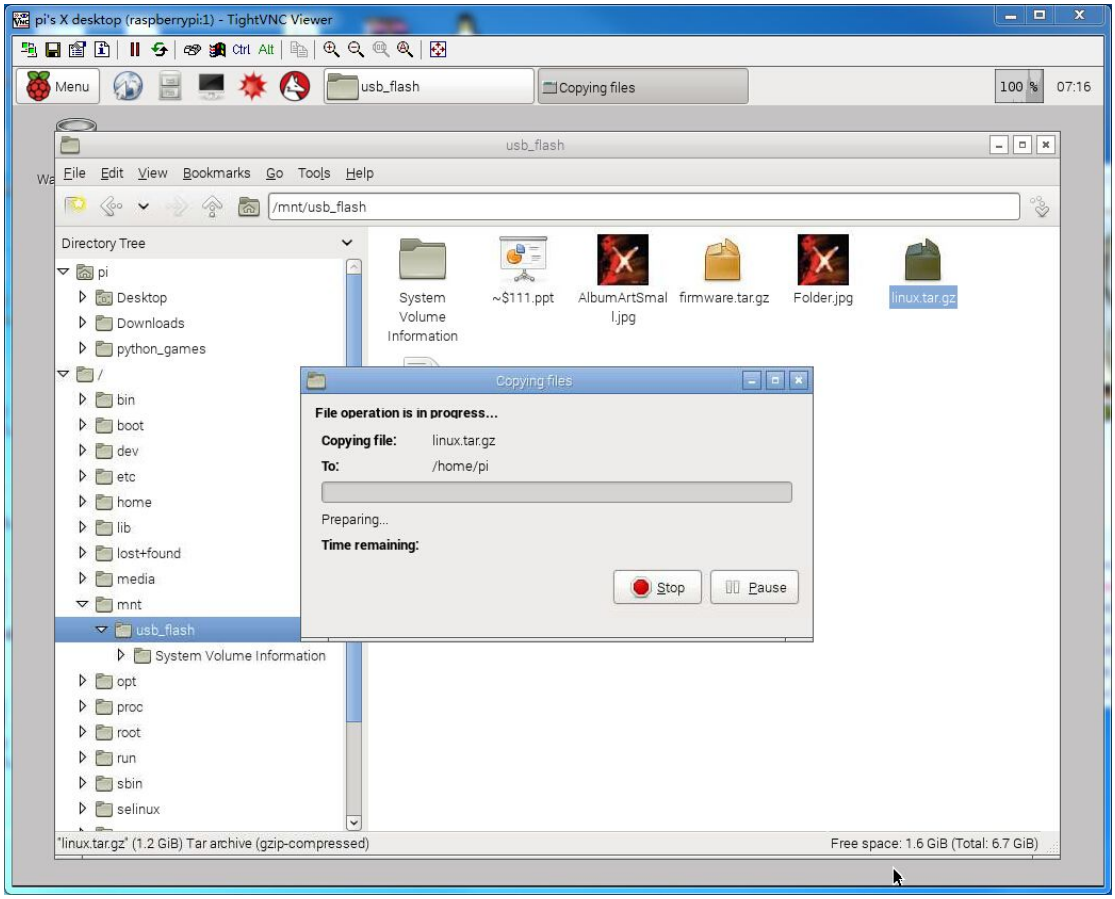
之后的事情更麻烦，首先，如何把 git 转移到树莓派中，我现在 linux 打包了 linux 和 firmware 文件夹，然后复制到宿主机中，但是 windows 却不认 EXT3\4 格式的文件系统，最后，我只能把打包的文件拷贝到 U 盘，树莓派再挂载 U 盘，拷贝文件到树莓派中。

挂载 U 盘：

```
$ sudo mount -o uid=pi,git=pi /dev/sda1 /mnt/usb_flash
```

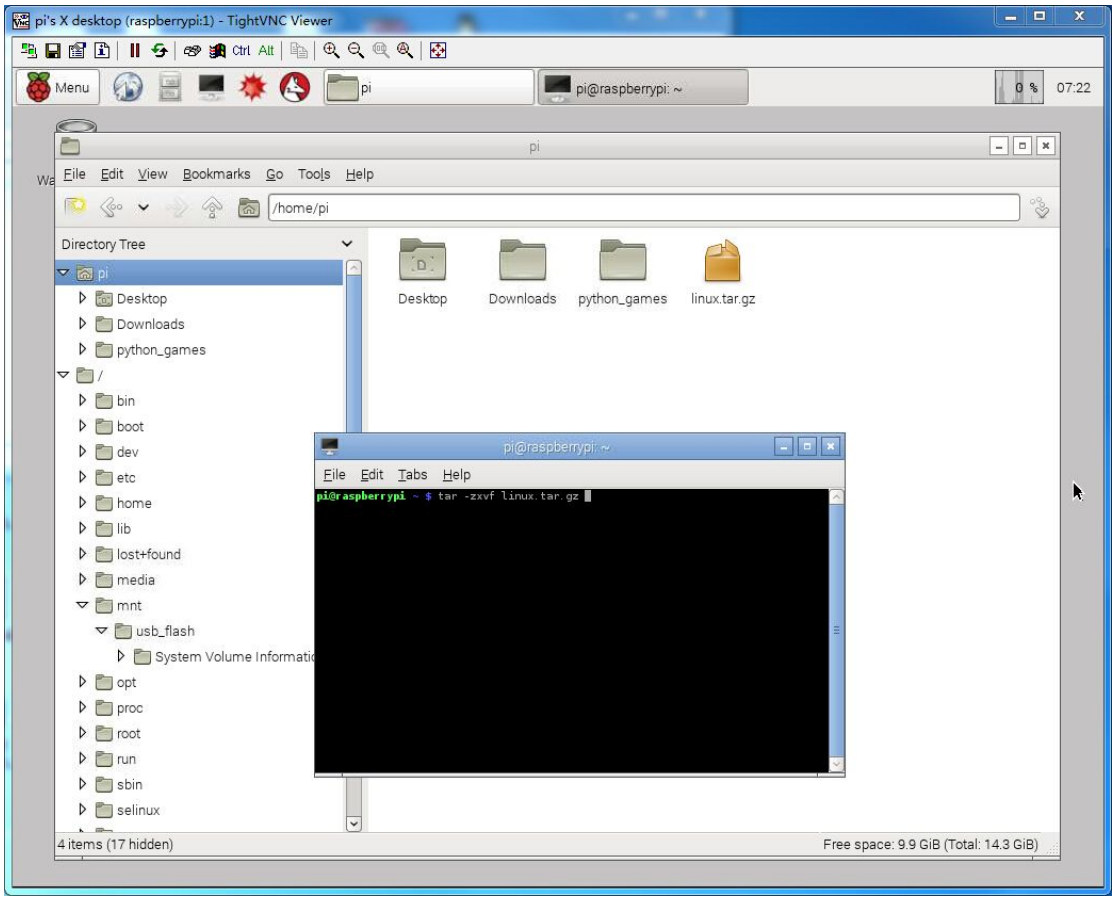


开始拷贝：



解包数据：

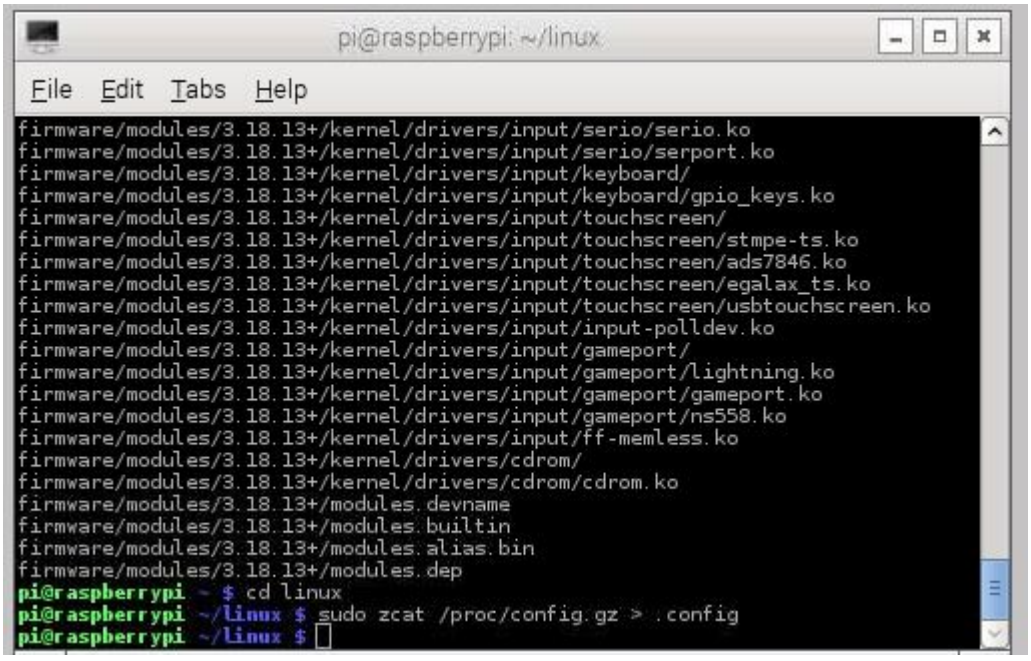
```
$ tar -zxvf linux.tar.gz
$ tar -zxvf firmware.tar.gz
```



2. 在内核中加入新的系统调用，具体功能没有要求，能输出调试信息即可

提取原配置文件：

```
$ cd linux
$ sudo zcat /proc/config.gz > .config
```

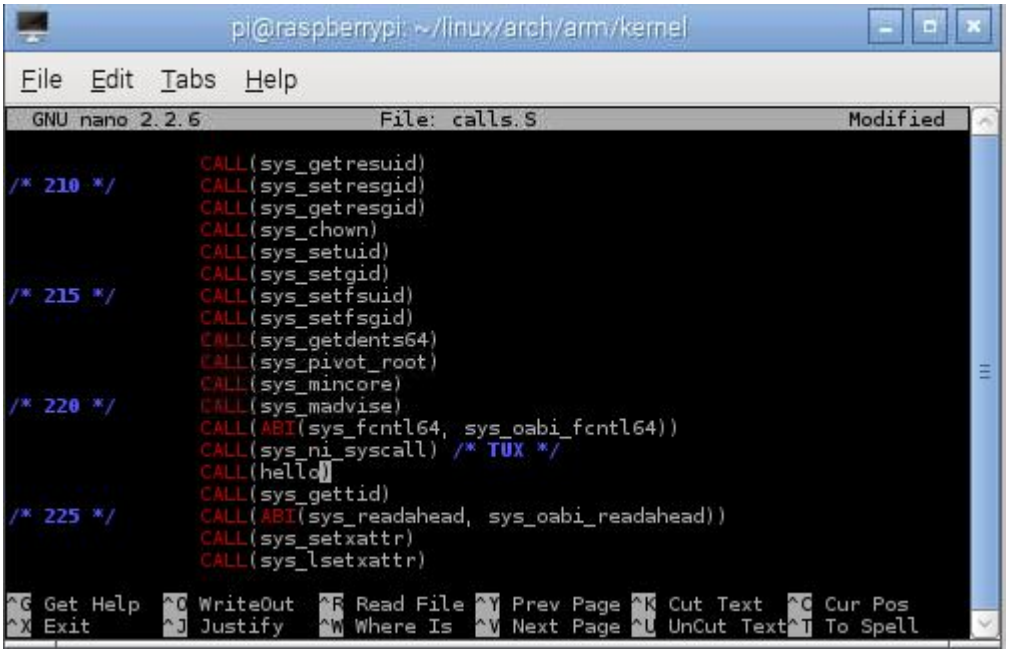


建立系统调用文件：

```
$ cd arch/arm/kernel
$ nano mysyscall.c
```

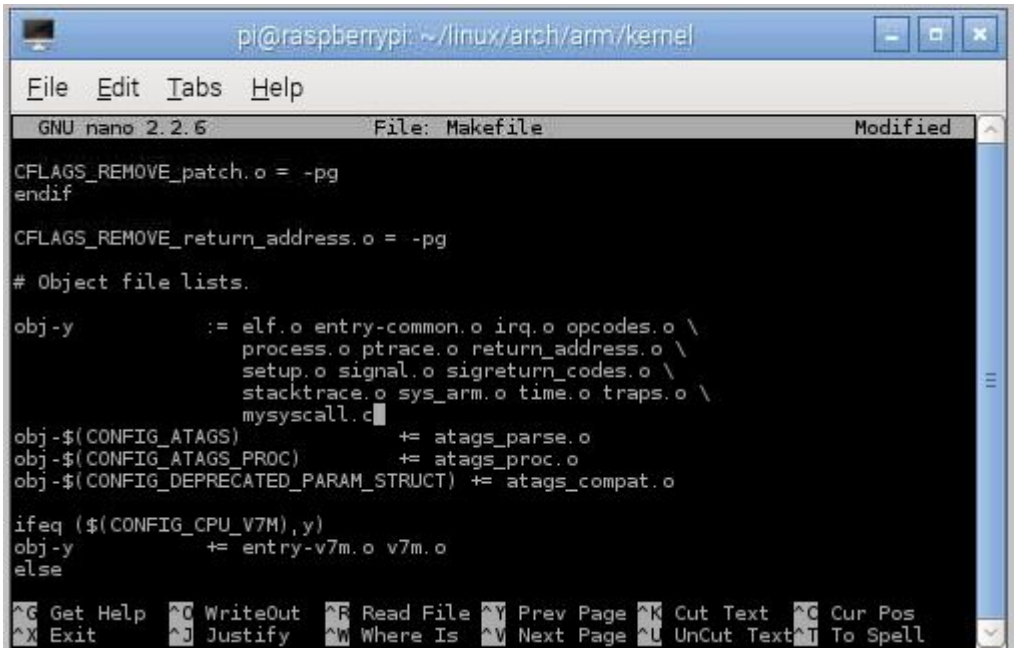
增加系统调用，将内核不使用的 223 号系统调用替换为新的系统调用：

```
$ nano calls.S
```



修改 Makefile 文件，增加编译 mysyscall.o：

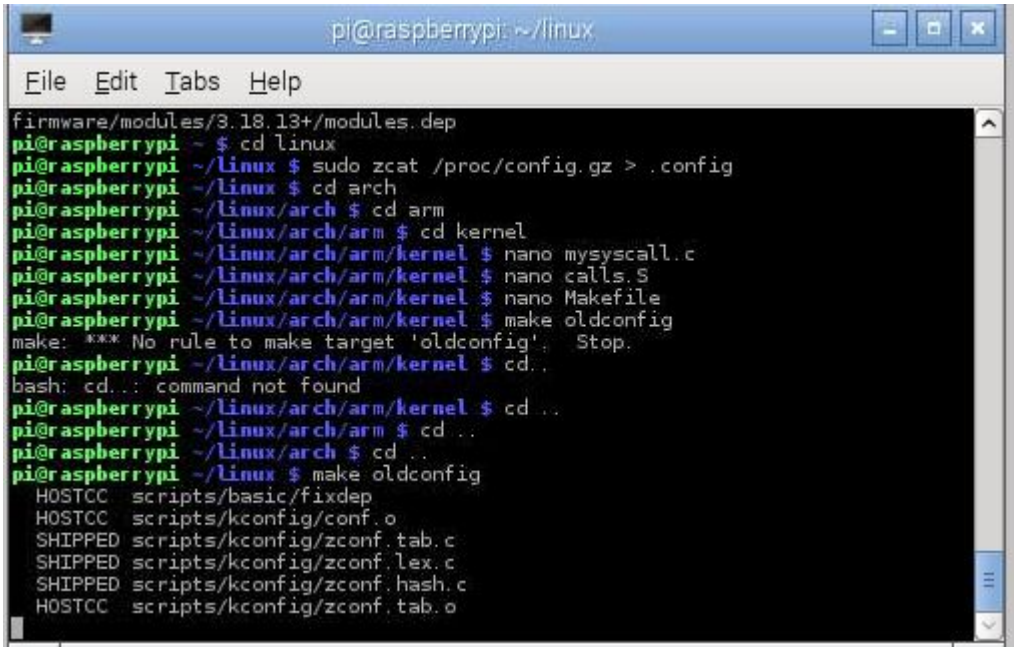
```
$ nano Makeflie
```



3. 修改内核代码配置，编译内核

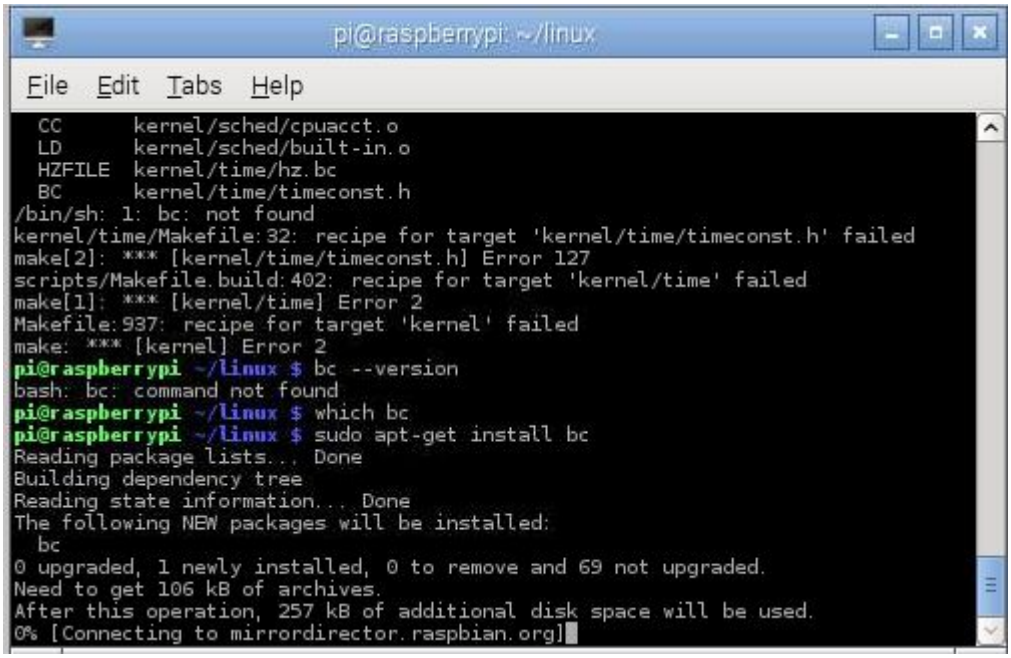
使用现有配置配置内核：

```
$ make oldconfig
```



在进行编译前一定要安装 bc，不然编译会出错，我编译了一晚上，第二天早上起来就发现这个错误：

```
$ sudo apt-get install bc
```



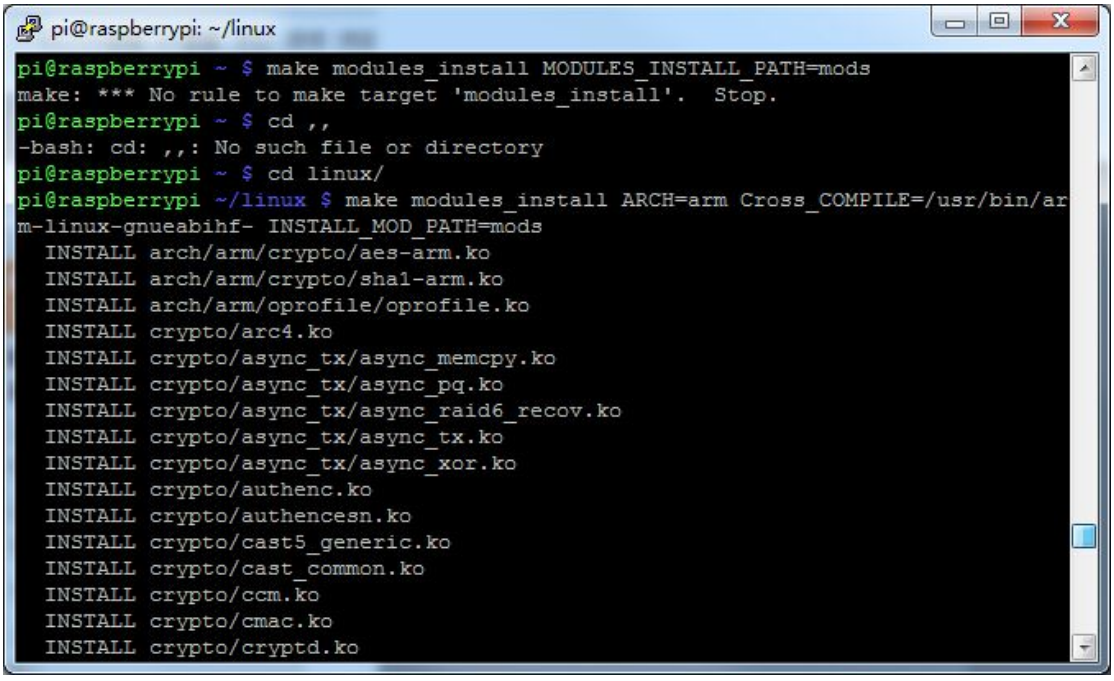
然后，执行：

```
$ make
```

整整 make 执行了 12 小时。。。。

编译模块：

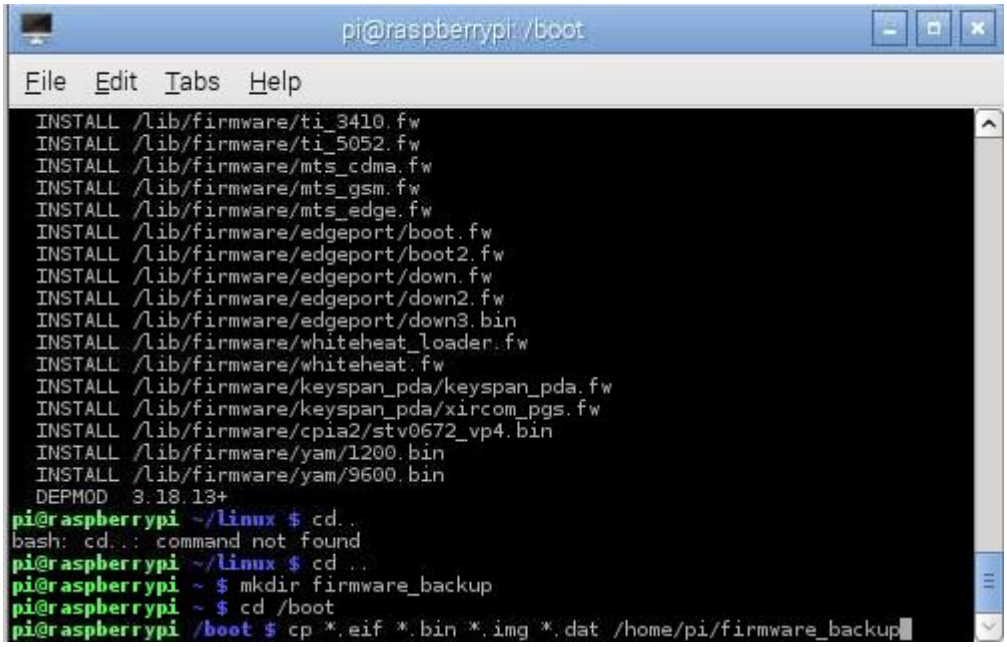
```
$ mkdir mods
$ make modules_install ARCH=arm
Cross-COMPILE=/usr/bin/arm-linux-gnueabinf- INSTALL_MOD_PATH=mods
```



4. 将编译好的内核装载到树莓派启动

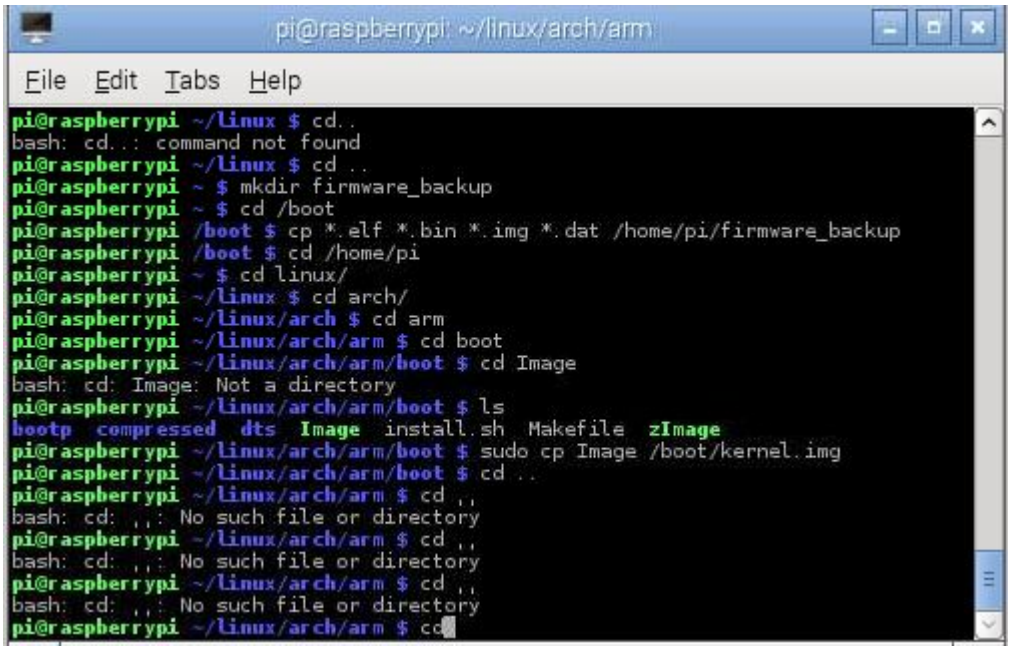
备份原内核和固件：

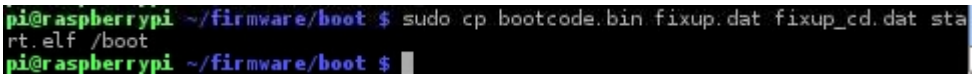
```
$ cd ..
$ mkdir firmware_backup
$ cd /boot
$ cp *.elf *.bin *.img *.dat /home/pi/firmware_backup
```



更新内核和固件：

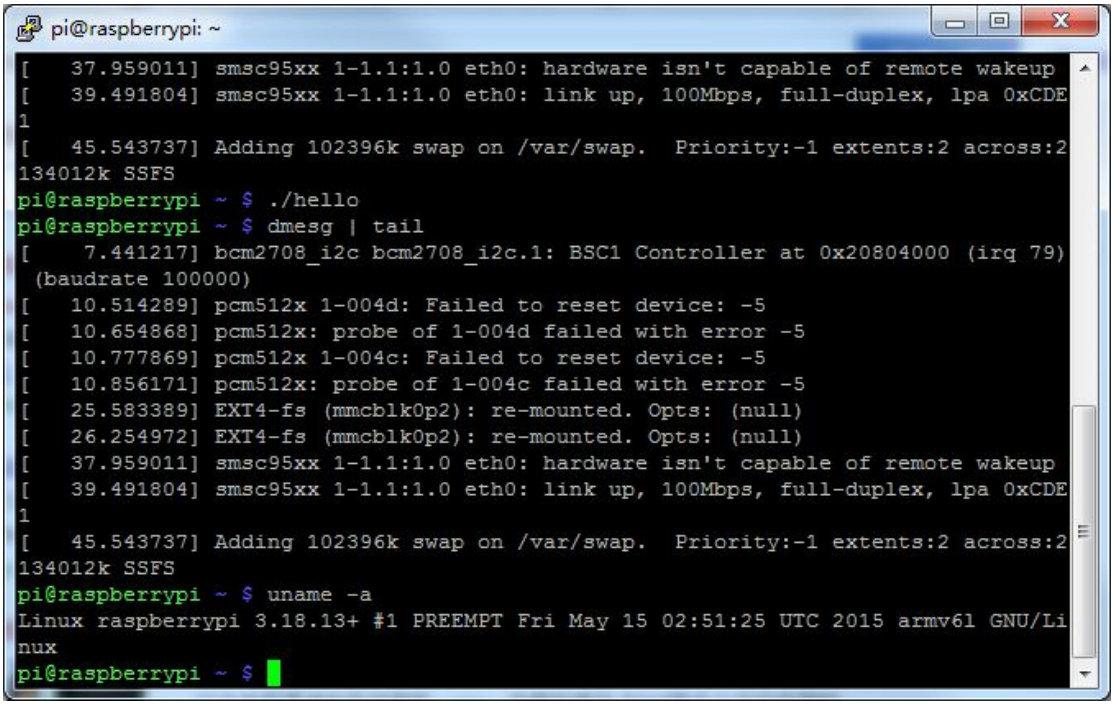
```
$ sudo cp /home/pi/linux/arch/arm/boot/Image /boot/kernel.img
$ sudo cp -r /home/pi/linux/mod/lib /
$ cd /home/pi/firmware/boot
$ sudo cp bootcode.bin fixup.dat fixup_cd.dat start.elf /boot
```





重启设备，并验证内核：

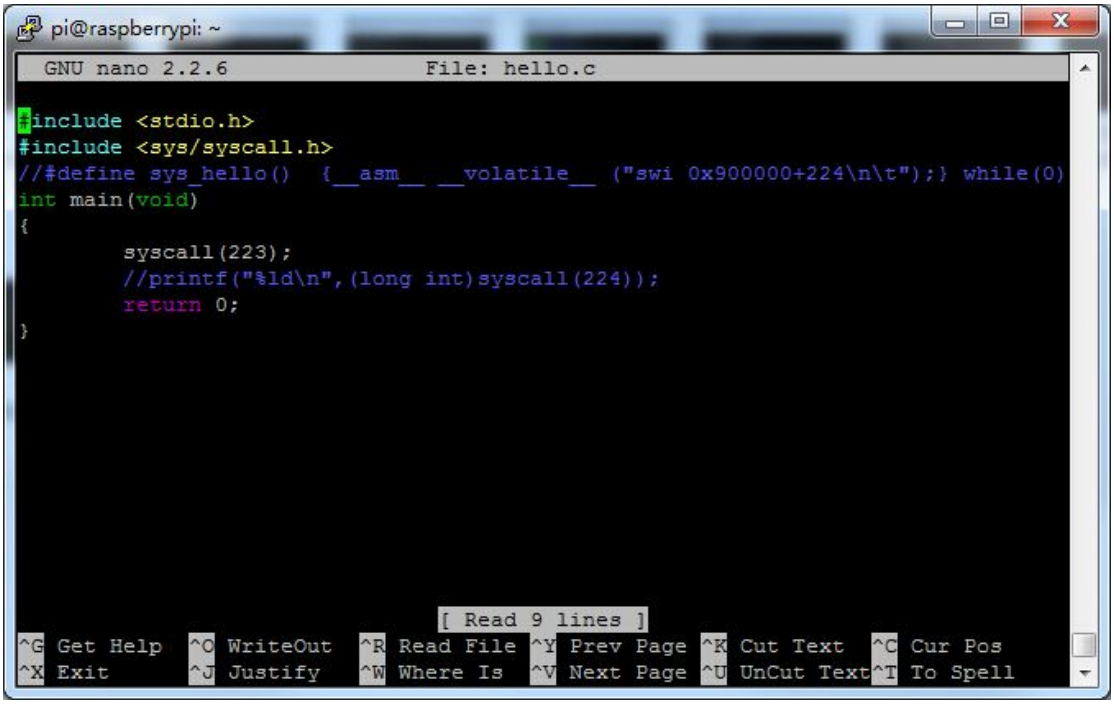
```
$ sudo reboot
$ uname -a
```



5. 编写 C 代码，用两种方法做系统调用，测试：

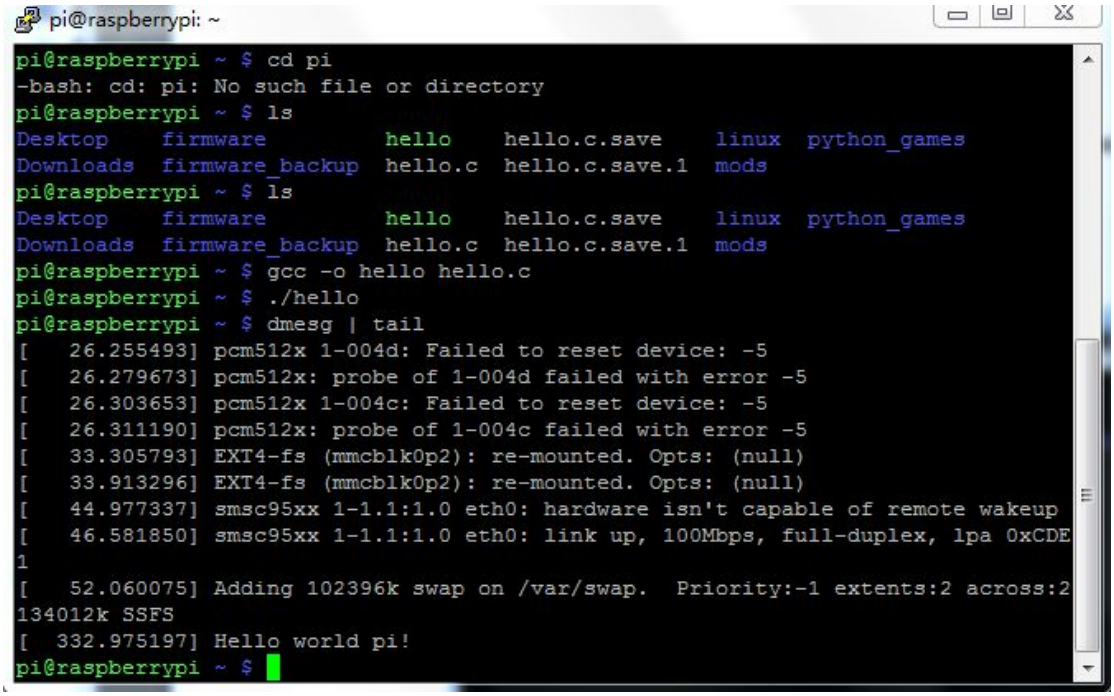
a) 用 syscall()函数

程序源码：



编译，后进行运行：

```
$ gcc -o hello hello.c
$ ./hello
$ dmesg | tail
```



b) 嵌入汇编代码，用 r0 传参数；

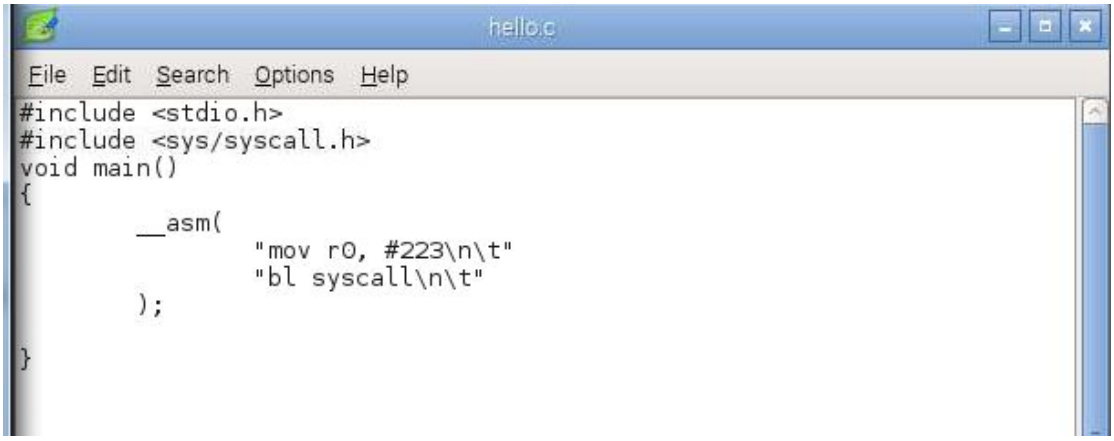
嵌入汇编代码对我来说，是第一接触，因此我在网上找了很多资料，其中走了不少弯路：

1. 树莓派的嵌入汇编代码是 ARM 的，不少 X86 的，其结构虽然大致相同，但寄存器名称和有些指令的使用是不同的

2. 光 ARM 的语法格式就有两种，一种是 ARM 开发工具编译环境下内嵌汇编语法格式，另一种是 GNU ARM 环境下内嵌汇编语法格式，GCC 可以编译的是 GNU ARM 环境下内嵌汇编语法格式

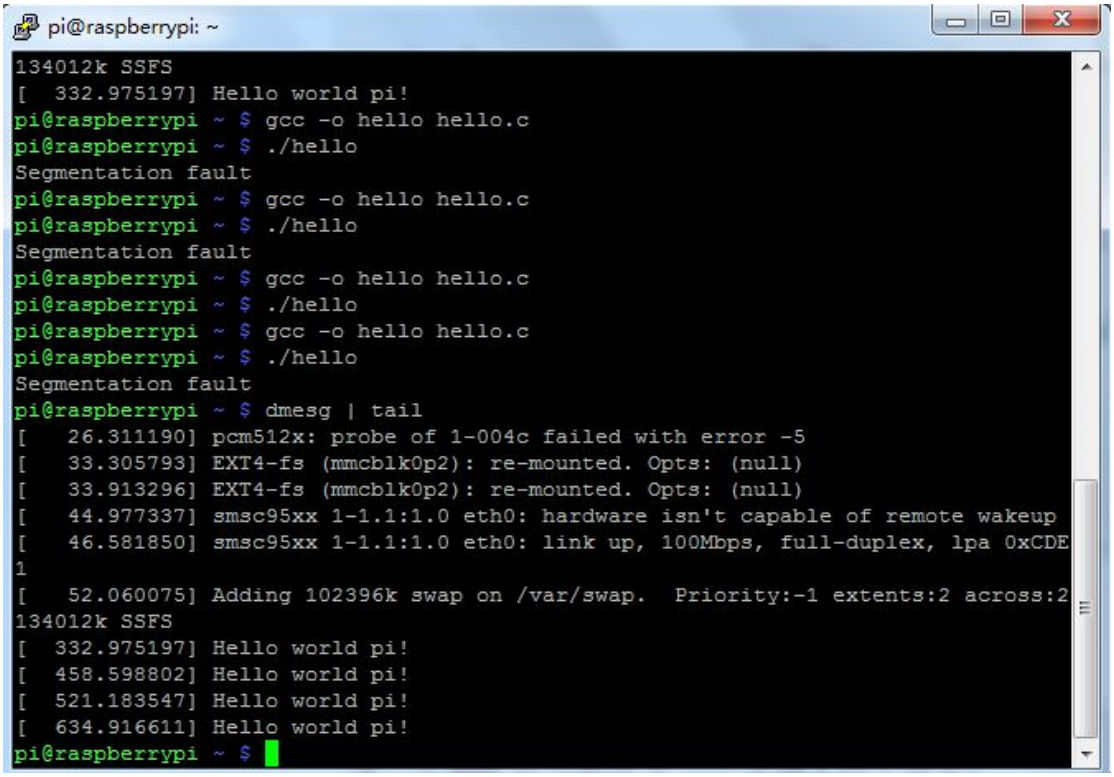
编写的过程也充满的艰辛，尝试了各种的编写过程，并研究了反汇编的代码，传入 R0 参数始终有问题，最后在网上查了各种资料，终于找到了嵌入汇编代码实现系统调用的方法。

源代码：



编译，后进行运行：

```
$ gcc -o hello hello.c
$ ./hello
$ dmesg | tail
```



由于我跑了三次嵌入汇编的系统调用，所以出现了 4 个”Hello world pi!”的调试信息，实验成功。

五、实验数据记录和处理

1. 寻找、下载 pcDuino 的 Linux 内核源码；
   花了两天，终于用虚拟机下载，U 盘拷贝的方式成功了
2. 在内核中加入新的系统调用，具体功能没有要求，能输出调试信息即可;
   成功
3. 修改内核代码配置，编译内核；
   被 BC 坑到，又编译了一天一夜才成功
4. 将编译好的内核装载到 pcDuino 启动；
   成功
5. 编写 C 代码，用两种方法做系统调用，测试：
   a) 嵌入汇编代码，用 r0 传参数；
      学习了嵌入汇编的内容，尝试了各种方式的编程，并尝试了 R0 - R3 的传入和 R0 的传出，也对照反汇编代码进行研究，最终成功完成了系统调用。
   b) 用 syscall()函数。
      成功。

六、实验结果与分析

实验成功。

七、讨论、心得

最困难的一次实验，首先是不稳定的 git 网站，然后是漫长的编译过程，最后是汇编代码的学习，虽然花了很多的时间和心思，但最后终于完成了实验。

另外我总结了一下在 x86 Linux 上做相同的系统调用，两者的不同，由于我使用的系统调用与我在操作系统实验所做的 x86 Linux 系统添加系统调用的代码相同，所有我发现其主要的区别在于

1. ARM 只需要修改 calls.S 和 Makefile 文件，X86 需要修改 unistd.h、syscall_table_32.S 和 Makefile 文件
2. 嵌入汇编的不同，主要在于寄存器名称和指令，ARM 有{cond}域，X86 则没有