

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。

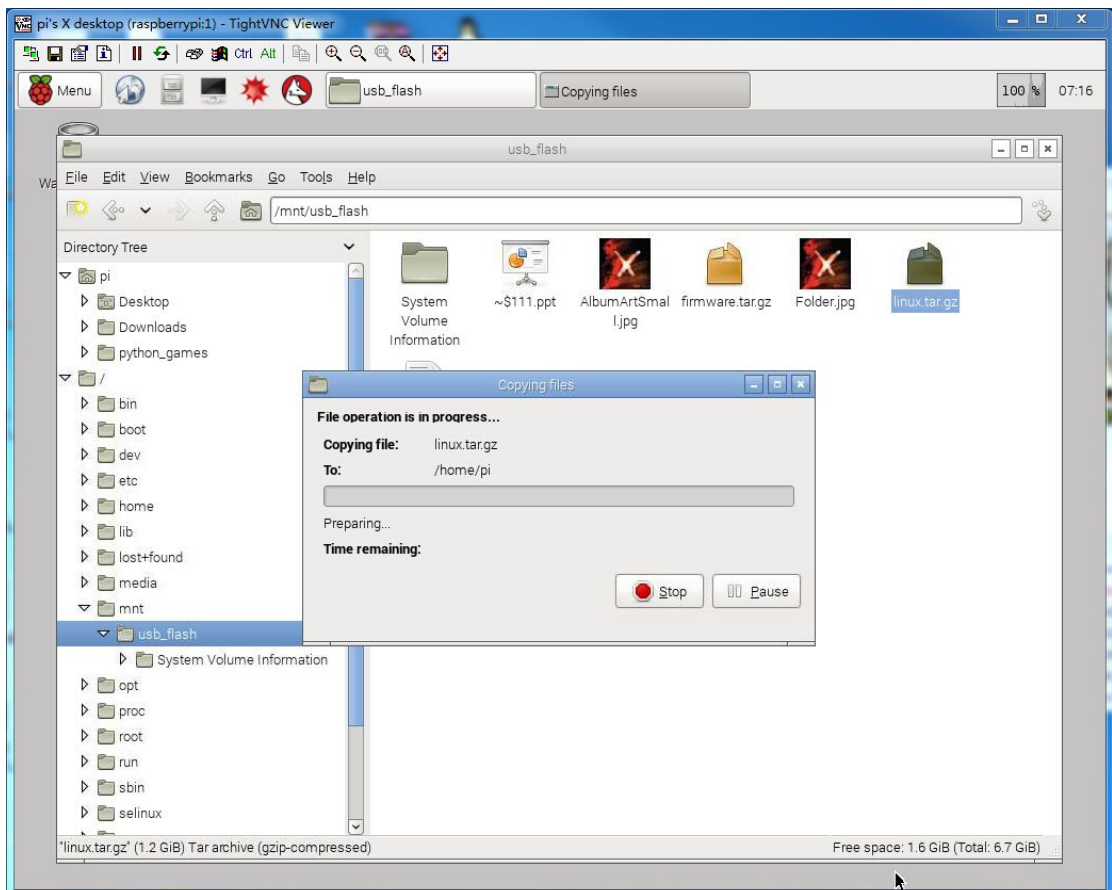
之后的事情更麻烦，首先，如何把 git 转移到树莓派中，我现在 linux 打包了 linux 和 firmware 文件夹，然后复制到宿主机中，但是 windows 却不认 EXT3\4 格式的文件系统，最后，我只能把打包的文件拷贝到 U 盘，树莓派再挂载 U 盘，拷贝文件到树莓派中。

挂载 U 盘：

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

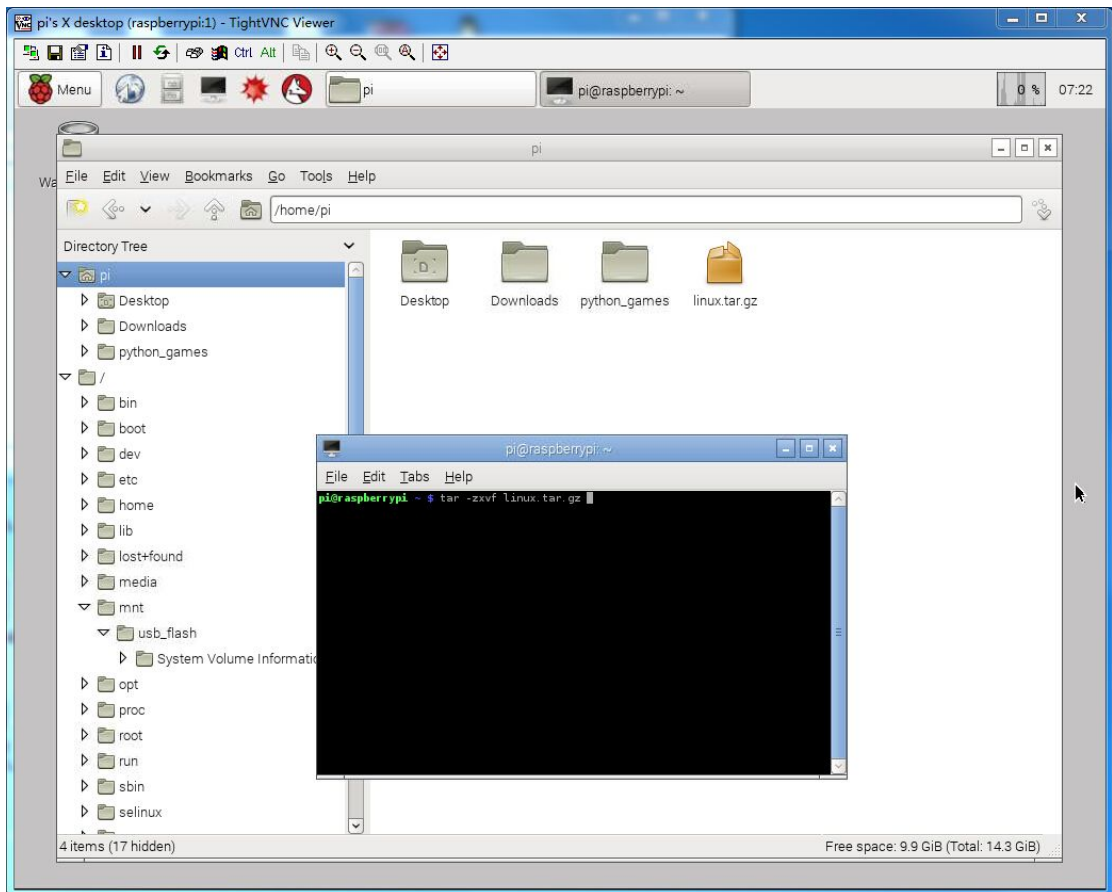
```
pi@raspberrypi: ~  
login as: pi  
pi@192.168.3.111's password:  
Linux raspberrypi 3.18.7+ #755 PREEMPT Thu Feb 12 17:14:31 GMT 2015 armv6l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Thu May 14 06:10:48 2015 from 192.168.3.237  
pi@raspberrypi ~ $ sudo mkdir /mnt/usb_flash  
pi@raspberrypi ~ $ mount -o uid=pi,gid=pi /dev/sda1 /mnt/usb_flash  
mount: only root can do that  
pi@raspberrypi ~ $ sudo mount -o uid=pi,gid=pi /dev/sda1 /mnt/usb_flash  
pi@raspberrypi ~ $
```

开始拷贝:



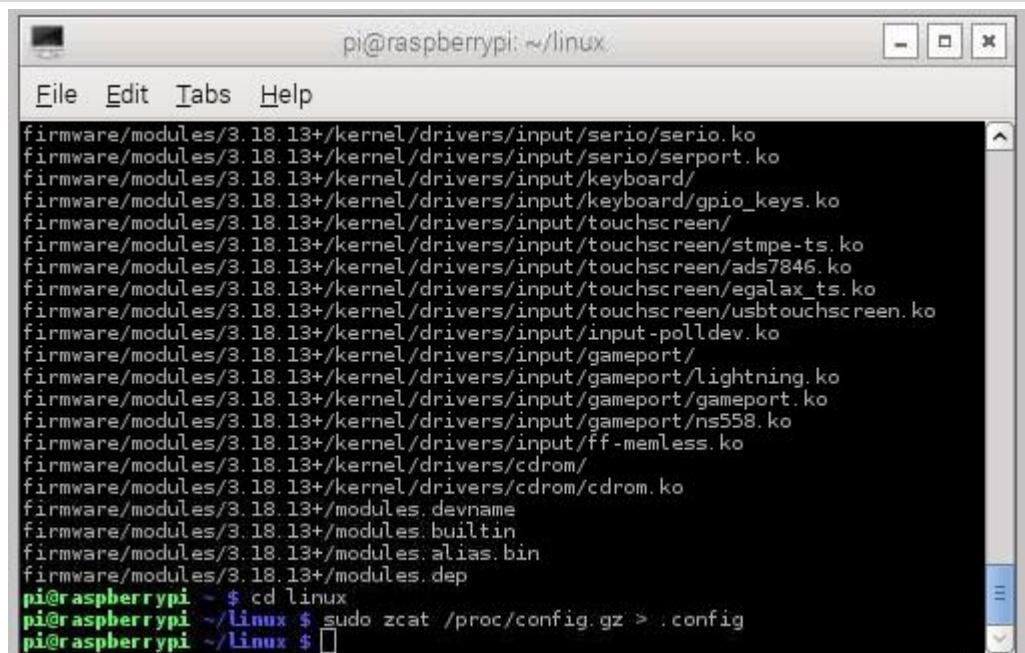
解包数据:

```
$ 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
```



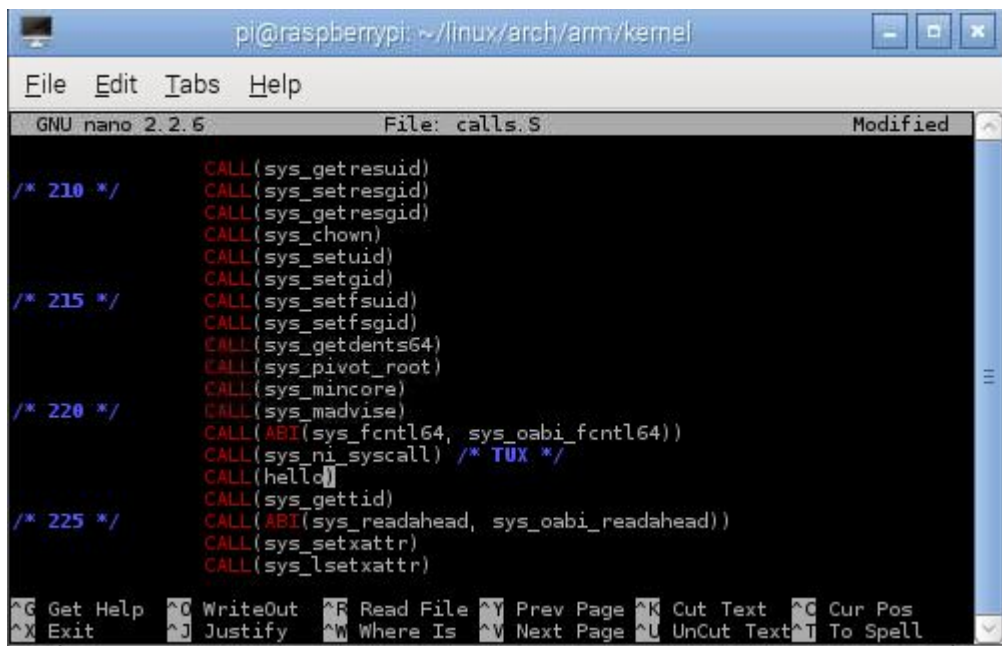
```
pi@raspberrypi: ~/linux/arch/arm/kernel
File Edit Tabs Help
GNU nano 2.2.6 File: mysyscall.c

#include<linux/kernel.h>
void hello(void)
{
    printk("Hello world pi!\n");
}

[ Wrote 5 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^L UnCut Text ^T To Spell
```

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
```



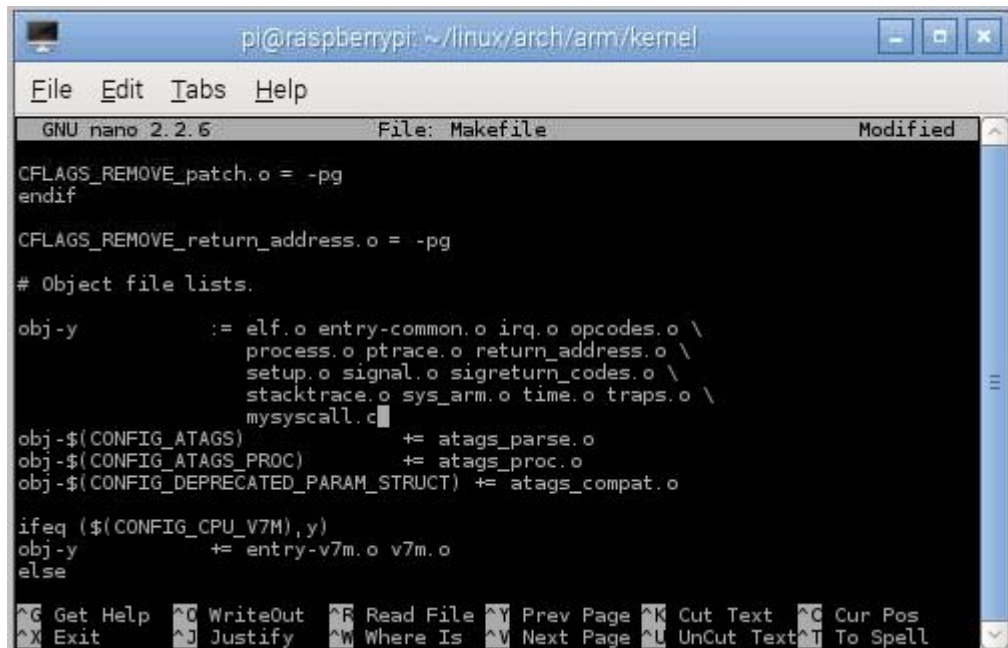
```
pi@raspberrypi: ~/linux/arch/arm/kernel
File Edit Tabs Help
GNU nano 2.2.6 File: calls.S Modified

/* 210 */    CALL(sys_getresuid)
              CALL(sys_setresgid)
              CALL(sys_getresgid)
              CALL(sys_chown)
              CALL(sys_setuid)
              CALL(sys_setgid)
/* 215 */    CALL(sys_setfsuid)
              CALL(sys_setfsgid)
              CALL(sys_getdents64)
              CALL(sys_pivot_root)
              CALL(sys_mincore)
/* 220 */    CALL(sys_madvise)
              CALL(ABI(sys_fcntl64, sys_oabi_fcntl64))
              CALL(sys_ni_syscall) /* TUX */
              CALL(hello)
              CALL(sys_gettid)
/* 225 */    CALL(ABI(sys_readahead, sys_oabi_readahead))
              CALL(sys_setxattr)
              CALL(sys_lsetxattr)

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^L UnCut Text ^T To Spell
```

修改 Makefile 文件，增加编译 mysyscall.o:

```
$ nano Makefile
```

```
pi@raspberrypi: ~/linux/arch/arm/kernel
File Edit Tabs Help
GNU nano 2.2.6 File: Makefile Modified
CFLAGS_REMOVE_patch.o = -pg
endif

CFLAGS_REMOVE_return_address.o = -pg

# Object file lists.

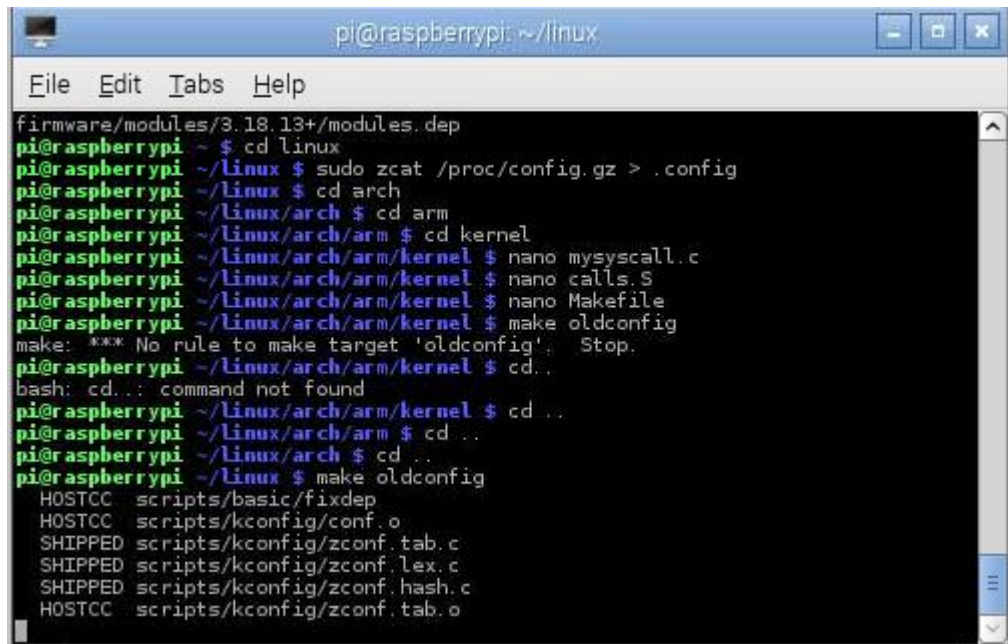
obj-y      := elf.o entry-common.o irq.o opcodes.o \
              process.o ptrace.o return_address.o \
              setup.o signal.o sigreturn_codes.o \
              stacktrace.o sys_arm.o time.o traps.o \
              msyscall.c
obj-$(CONFIG_ATAGS) += atags_parse.o
obj-$(CONFIG_ATAGS_PROC) += atags_proc.o
obj-$(CONFIG_DEPRECATED_PARAM_STRUCT) += atags_compat.o

ifeq ($(CONFIG_CPU_V7M),y)
obj-y      += entry-v7m.o v7m.o
else
```

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
```



```
pi@raspberrypi: ~/linux
File Edit Tabs Help
firmware/modules/3.18.13+/modules.dep
pi@raspberrypi ~ $ cd linux
pi@raspberrypi ~/linux $ sudo zcat /proc/config.gz > .config
pi@raspberrypi ~/linux $ cd arch
pi@raspberrypi ~/linux/arch $ cd arm
pi@raspberrypi ~/linux/arch/arm $ cd kernel
pi@raspberrypi ~/linux/arch/arm/kernel $ nano msyscall.c
pi@raspberrypi ~/linux/arch/arm/kernel $ nano calls.S
pi@raspberrypi ~/linux/arch/arm/kernel $ nano Makefile
pi@raspberrypi ~/linux/arch/arm/kernel $ make oldconfig
make: *** No rule to make target 'oldconfig'. Stop.
pi@raspberrypi ~/linux/arch/arm/kernel $ cd ..
bash: cd..: command not found
pi@raspberrypi ~/linux/arch/arm/kernel $ cd ..
pi@raspberrypi ~/linux/arch/arm $ cd ..
pi@raspberrypi ~/linux/arch $ cd ..
pi@raspberrypi ~/linux $ make oldconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
SHIPPED scripts/kconfig/zconf.tab.c
SHIPPED scripts/kconfig/zconf.lex.c
SHIPPED scripts/kconfig/zconf.hash.c
HOSTCC scripts/kconfig/zconf.tab.o
```

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
```

```
pi@raspberrypi: ~/linux
File Edit Tabs Help
CC      kernel/sched/cpuacct.o
LD      kernel/sched/built-in.o
HZFILE  kernel/time/hz.bc
BC      kernel/time/timeconst.h
/bin/sh: 1: bc: not found
kernel/time/Makefile:32: recipe for target 'kernel/time/timeconst.h' failed
make[2]: *** [kernel/time/timeconst.h] Error 127
scripts/Makefile.build:402: recipe for target 'kernel/time' failed
make[1]: *** [kernel/time] Error 2
Makefile:937: recipe for target 'kernel' failed
make: *** [kernel] Error 2
pi@raspberrypi ~/linux $ bc --version
bash: bc: command not found
pi@raspberrypi ~/linux $ which bc
pi@raspberrypi ~/linux $ sudo apt-get install bc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  bc
0 upgraded, 1 newly installed, 0 to remove and 69 not upgraded.
Need to get 106 kB of archives.
After this operation, 257 kB of additional disk space will be used.
0% [Connecting to mirror.director.raspbian.org]
```

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-gnueabihf- INSTALL_MOD_PATH=mods
```

```
pi@raspberrypi: ~/linux
pi@raspberrypi ~ $ make modules_install MODULES_INSTALL_PATH=mods
make: *** No rule to make target 'modules_install'. Stop.
pi@raspberrypi ~ $ cd ..
-bash: cd: ..: No such file or directory
pi@raspberrypi ~ $ cd linux/
pi@raspberrypi ~/linux $ make modules_install ARCH=arm Cross_COMPILE=/usr/bin/arm-linux-gnueabihf- INSTALL_MOD_PATH=mods
INSTALL arch/arm/crypto/aes-arm.ko
INSTALL arch/arm/crypto/sha1-arm.ko
INSTALL arch/arm/oprofile/oprofile.ko
INSTALL crypto/arc4.ko
INSTALL crypto/async_tx/async_memcpy.ko
INSTALL crypto/async_tx/async_pq.ko
INSTALL crypto/async_tx/async_raid6_recov.ko
INSTALL crypto/async_tx/async_tx.ko
INSTALL crypto/async_tx/async_xor.ko
INSTALL crypto/authenc.ko
INSTALL crypto/authencsn.ko
INSTALL crypto/cast5_generic.ko
INSTALL crypto/cast_common.ko
INSTALL crypto/ccm.ko
INSTALL crypto/cmac.ko
INSTALL crypto/cryptd.ko
```

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
```



```
pi@raspberrypi: ~  
[ 37.959011] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup  
[ 39.491804] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE  
1  
[ 45.543737] Adding 102396k swap on /var/swap. Priority:-1 extents:2 across:2  
134012k SSFS  
pi@raspberrypi ~ $ ./hello  
pi@raspberrypi ~ $ dmesg | tail  
[ 7.441217] bcm2708_i2c bcm2708_i2c.1: BSC1 Controller at 0x20804000 (irq 79)  
(baudrate 100000)  
[ 10.514289] pcm512x 1-004d: Failed to reset device: -5  
[ 10.654868] pcm512x: probe of 1-004d failed with error -5  
[ 10.777869] pcm512x 1-004c: Failed to reset device: -5  
[ 10.856171] pcm512x: probe of 1-004c failed with error -5  
[ 25.583389] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 26.254972] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 37.959011] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup  
[ 39.491804] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE  
1  
[ 45.543737] Adding 102396k swap on /var/swap. Priority:-1 extents:2 across:2  
134012k SSFS  
pi@raspberrypi ~ $ uname -a  
Linux raspberrypi 3.18.13+ #1 PREEMPT Fri May 15 02:51:25 UTC 2015 armv6l GNU/Li  
nux  
pi@raspberrypi ~ $
```

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：

```
GNU nano 2.2.6 File: hello.c  
#include <stdio.h>  
#include <sys/syscall.h>  
//#define sys_hello() {__asm__ __volatile__ ("swi 0x9000000+224\n\t");} while(0)  
int main(void)  
{  
    syscall(223);  
    //printf("%ld\n", (long int)syscall(224));  
    return 0;  
}  
[ Read 9 lines ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

编译，后进行运行：

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

```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ cd pi  
-bash: cd: pi: No such file or directory  
pi@raspberrypi ~ $ ls  
Desktop    firmware      hello    hello.c.save  linux  python_games  
Downloads  firmware_backup hello.c    hello.c.save.1  mods  
pi@raspberrypi ~ $ ls  
Desktop    firmware      hello    hello.c.save  linux  python_games  
Downloads  firmware_backup hello.c    hello.c.save.1  mods  
pi@raspberrypi ~ $ gcc -o hello hello.c  
pi@raspberrypi ~ $ ./hello  
pi@raspberrypi ~ $ dmesg | tail  
[ 26.255493] pcm512x 1-004d: Failed to reset device: -5  
[ 26.279673] pcm512x: probe of 1-004d failed with error -5  
[ 26.303653] pcm512x 1-004c: Failed to reset device: -5  
[ 26.311190] pcm512x: probe of 1-004c failed with error -5  
[ 33.305793] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 33.913296] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 44.977337] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup  
[ 46.581850] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE  
1  
[ 52.060075] Adding 102396k swap on /var/swap. Priority:-1 extents:2 across:2  
134012k SSFS  
[ 332.975197] Hello world pi!  
pi@raspberrypi ~ $
```

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：

```
hello.c  
File Edit Search Options Help  
#include <stdio.h>  
#include <sys/syscall.h>  
void main()  
{  
    __asm(  
        "mov r0, #223\n\t"  
        "bl syscall\n\t"  
    );  
}
```

编译，后进行运行：

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

```
pi@raspberrypi: ~  
134012k SSFS  
[ 332.975197] Hello world pi!  
pi@raspberrypi ~ $ gcc -o hello hello.c  
pi@raspberrypi ~ $ ./hello  
Segmentation fault  
pi@raspberrypi ~ $ gcc -o hello hello.c  
pi@raspberrypi ~ $ ./hello  
Segmentation fault  
pi@raspberrypi ~ $ gcc -o hello hello.c  
pi@raspberrypi ~ $ ./hello  
pi@raspberrypi ~ $ gcc -o hello hello.c  
pi@raspberrypi ~ $ ./hello  
Segmentation fault  
pi@raspberrypi ~ $ dmesg | tail  
[ 26.311190] pcm512x: probe of 1-004c failed with error -5  
[ 33.305793] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 33.913296] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)  
[ 44.977337] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup  
[ 46.581850] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE  
1  
[ 52.060075] Adding 102396k swap on /var/swap. Priority:-1 extents:2 across:2  
134012k SSFS  
[ 332.975197] Hello world pi!  
[ 458.598802] Hello world pi!  
[ 521.183547] Hello world pi!  
[ 634.916611] Hello world pi!  
pi@raspberrypi ~ $
```

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，所以我发现主要的区别在于

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