

# 浙江大学

## 本科实验报告

课程名称: 嵌入式系统

姓 名: Jessie Peng

学 院: 计算机科学与技术学院

系:

专 业:

学 号:

指导教师: 王总辉

2019 年 6 月 23 日

# 浙江大学实验报告

课程名称： 嵌入式系统 实验类型： 综合

实验项目名称： 实验 6：嵌入式 Linux 内核

学生姓名： Jessie Peng 专业：  学号：

实验地点： 曹西 501 实验日期： 2019 年 6 月 23 日 指导老师： 王总辉

## 一、实验目的和要求

- 学习 Linux 内核的配置和编译
- 理解 ARM 和 x86 的 CPU 模式（系统模式、用户模式等）的不同
- 深入理解 Linux 系统调用，掌握编写自定义 Linux 系统调用的方法
- 掌握内核模块的编写方法

## 二、实验内容和原理

### 【实验器材——硬件】

- 树莓派 3B+实验板
- 5V/1A 电源
- microUSB 线
- 笔记本电脑
- HDMI 线、HDMI 显示器

### 【实验器材——软件】

- PC 上的 SSH 软件
- 交叉编译软件 arm-linux-gcc

### 【实验内容——增加自定义系统调用】

- 寻找、下载树莓派所用的 Linux 内核源码

- 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
- 修改内核代码配置，编译内核
- 将编译好的内核装载到板卡启动
- 编写 C 代码，用两种方法做系统调用，测试
  - 嵌入汇编代码，用 r0 传参数
  - 用 syscall ( ) 函数

#### 【实验内容——增加内核模块】

- 编写内核模块，在模块加载和卸载时能通过内核打印函数输出提示信息
- 通过 insmod 和 lsmod 等命令测试内核模块

### 三、实验步骤和结果记录

#### 【树莓派 Linux 内核编译】

在正式开始实验内容之前，先选择一个合适的内核版本，在 Ubuntu 虚拟机上进行交叉编译，然后下载到树莓派上尝试运行。根据官方文档，本次实验的内核编译与安装遵循以下步骤：

#### 1. 下载交叉编译工具链

```
$ git clone https://github.com/raspberrypi/tools ~/tools
```

#### 2. 设置环境变量

```
$ echo
PATH=\$PATH:~/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin >> ~/.bashrc
source ~/.bashrc
```

#### 3. 下载内核源码

```
$ git clone --depth=1 --branch rpi-4.9.y
https://github.com/raspberrypi/linux linux-4.9
```

#### 4. 安装依赖

```
$ sudo apt-get install git bison flex libssl-dev
```

## 5. 编译源码自带的配置文件

```
$ cd linux-4.9  
$ KERNEL=kernel7  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
```

## 6. 编译内核

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules  
dtbs -j 4
```

## 7. 挂载 SD 卡的 boot 和根文件系统分区

```
$ mkdir mnt  
$ mkdir mnt/fat32  
$ mkdir mnt/ext4  
$ sudo mount /dev/sdb1 mnt/fat32  
$ sudo mount /dev/sdb2 mnt/ext4
```

## 8. 安装根文件系统模块

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
INSTALL_MOD_PATH=mnt/ext4 modules_install
```

## 9. 备份 SD 卡上旧的内核映像

```
$ sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
```

## 10. 拷贝新的内核映像和设备树到 SD 卡上

```
$ sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img  
$ sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/  
$ sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/  
$ sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
```

## 11. 卸载 SD 卡

```
$ sudo umount mnt/fat32
$ sudo umount mnt/ext4
```

最后将 SD 卡插到树莓派上，上电运行即可，如果运行失败，考虑换一个版本的内核源码，重新编译。

### 【增加自定义系统调用】

在 linux-4.9/arch/arm/kernel/目录下增加自己的系统调用文件 sys\_mysyscall.c:

```
1. #include <linux/kernel.h>
2.
3. void sys_mysyscall(void) {
4.     printk("This is my system call for Lab6\n");
5. }
```

修改同一目录下的 Makefile，增加 sys\_mysyscall.o 目标:

```
1. # Object file lists.
2.
3. obj-y      := elf.o entry-common.o irq.o opcodes.o \
4.               process.o ptrace.o reboot.o return_address.o \
5.               setup.o signal.o sigreturn_codes.o \
6.               stacktrace.o sys_arm.o time.o traps.o \
7.               sys_mysyscall.o
```

修改同一目录下的系统中断向量表，替换掉没有用的 223 号:

```
1. /* 220 */ CALL(sys_madvise)
2. CALL(ABI(sys_fcntl64, sys_oabi_fcntl64))
3. CALL(sys_ni_syscall) /* TUX */
4. CALL(sys_mysyscall) /* my system call */
5. CALL(sys_gettid)
```

在头文件 linux-4.9/include/uapi/asm-generic/unistd.h 中修改 223 号系统调用指针声明:

```
1. /* mm/fadvise.c */
2. // #define __NR3264_fadvise64 223
3. // __SC_COMP(__NR3264_fadvise64, sys_fadvise64_64, compat_sys_fadvise64_64)
4. #define __NR3264_fadvise64 223
```

```
5. __SYSCALL(__NR_mysyscall, sys_mysyscall)
```

最后编译一下内核，系统调用就添加成功了。

### 【增加内核模块】

内核模块仍然在 Ubuntu 中交叉编译。

编写一个简单的内核模块，在加载时打印“Hello World”，退出时打印“Goodbye World”：

```
1. #include <linux/module.h>
2. #include <linux/kernel.h>
3. #include <linux/init.h>
4.
5. MODULE_LICENSE("GPL");
6. MODULE_AUTHOR("Jessie Peng <jessie_p@yeah.net>");
7. MODULE_DESCRIPTION("Kernel module for Lab6");
8.
9. static int __init hello(void)
10. {
11.     printk(KERN_INFO "Hello World\n");
12.     return 0;
13. }
14.
15. static void __exit goodbye(void)
16. {
17.     printk(KERN_INFO "Goodbye World\n");
18. }
19.
20. module_init(hello);
21. module_exit(goodbye);
```

编写 Makefile 进行交叉编译：

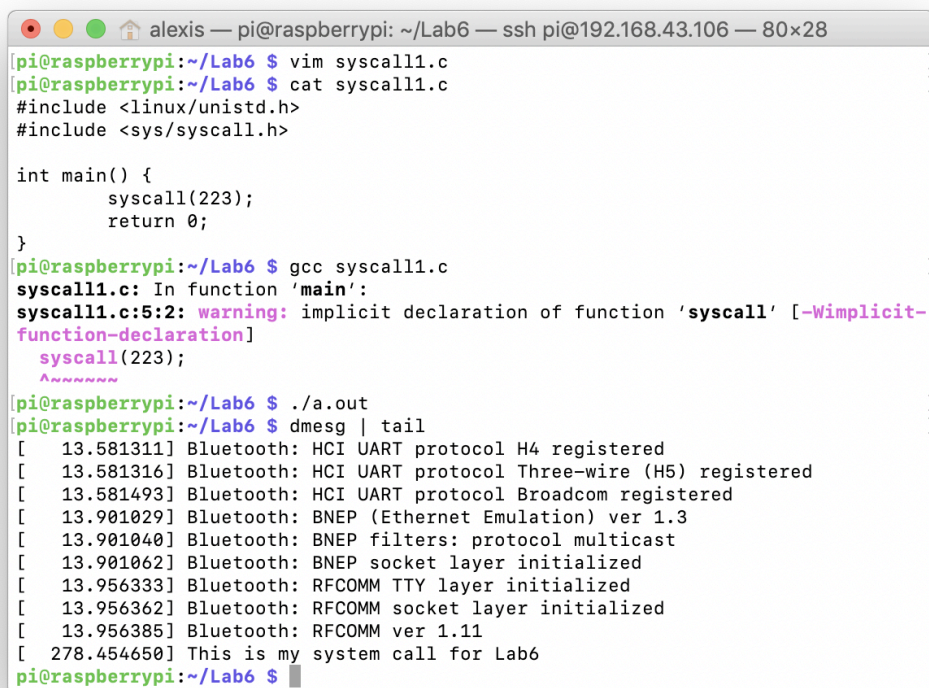
```
1. obj-m += hello.o
2.
3. PWD := $(shell pwd)
4. ARGS := ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
5. KERNEL_DIR := $(PWD)/../linux-4.9/
6.
7. all:
8.     make $(ARGS) -C $(KERNEL_DIR) SUBDIRS=$(PWD) modules
```

```
9. clean:
10.     rm *.o *.ko *.mod.c
11.
12. .PHONY: clean
```

生成的 hello.ko 即是编译好的内核模块。

## 四、实验结果分析

编写 syscall1.c, 使用 syscall()函数来调用 sys\_mysyscall。由于 KERN\_INFO 提示信息的优先级较低, 不会直接打印出来, 需要使用 smesg 命令查看日志消息。测试函数及结果如下图:



```
pi@raspberrypi: ~/Lab6 — ssh pi@192.168.43.106 — 80x28
pi@raspberrypi:~/Lab6 $ vim syscall1.c
pi@raspberrypi:~/Lab6 $ cat syscall1.c
#include <linux/unistd.h>
#include <sys/syscall.h>

int main() {
    syscall(223);
    return 0;
}
pi@raspberrypi:~/Lab6 $ gcc syscall1.c
syscall1.c: In function 'main':
syscall1.c:5:2: warning: implicit declaration of function 'syscall' [-Wimplicit-
function-declaration]
    syscall(223);
    ^~~~~~
pi@raspberrypi:~/Lab6 $ ./a.out
pi@raspberrypi:~/Lab6 $ dmesg | tail
[ 13.581311] Bluetooth: HCI UART protocol H4 registered
[ 13.581316] Bluetooth: HCI UART protocol Three-wire (H5) registered
[ 13.581493] Bluetooth: HCI UART protocol Broadcom registered
[ 13.901029] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 13.901040] Bluetooth: BNEP filters: protocol multicast
[ 13.901062] Bluetooth: BNEP socket layer initialized
[ 13.956333] Bluetooth: RFCOMM TTY layer initialized
[ 13.956362] Bluetooth: RFCOMM socket layer initialized
[ 13.956385] Bluetooth: RFCOMM ver 1.11
[ 278.454650] This is my system call for Lab6
pi@raspberrypi:~/Lab6 $
```

编写 syscall2.c, 嵌入汇编代码来使用系统调用, 测试函数及结果如下图:

```
alexis — pi@raspberrypi: ~/Lab6 — ssh pi@192.168.43.106 — 80x23
pi@raspberrypi:~/Lab6 $ vim syscall2.c
pi@raspberrypi:~/Lab6 $ cat syscall2.c
#include <stdio.h>
#define sys_call() {__asm__ __volatile__ ("swi 0x900000+223\n\t");} while(0)

int main(void) {
    sys_call();
    return 0;
}
pi@raspberrypi:~/Lab6 $ gcc syscall2.c
pi@raspberrypi:~/Lab6 $ ./a.out
pi@raspberrypi:~/Lab6 $ dmesg | tail
[ 13.581311] Bluetooth: HCI UART protocol H4 registered
[ 13.581316] Bluetooth: HCI UART protocol Three-wire (H5) registered
[ 13.581493] Bluetooth: HCI UART protocol Broadcom registered
[ 13.901029] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 13.901040] Bluetooth: BNEP filters: protocol multicast
[ 13.901062] Bluetooth: BNEP socket layer initialized
[ 13.956333] Bluetooth: RFCOMM TTY layer initialized
[ 13.956362] Bluetooth: RFCOMM socket layer initialized
[ 13.956385] Bluetooth: RFCOMM ver 1.11
[ 278.454650] This is my system call for Lab6
pi@raspberrypi:~/Lab6 $
```

使用 `insmod hello.ko` 命令加载内核模块，使用 `rmmod hello` 命令卸载内核模块，分别查看提示信息，如下图：

```
alexis — pi@raspberrypi: ~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules...
pi@raspberrypi:~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules $ sudo insmod hello.ko
pi@raspberrypi:~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules $ dmesg | tail
[ 12.951744] Bluetooth: HCI UART protocol Three-wire (H5) registered
[ 12.951840] Bluetooth: HCI UART protocol Broadcom registered
[ 13.276948] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 13.276961] Bluetooth: BNEP filters: protocol multicast
[ 13.276985] Bluetooth: BNEP socket layer initialized
[ 13.343554] Bluetooth: RFCOMM TTY layer initialized
[ 13.343586] Bluetooth: RFCOMM socket layer initialized
[ 13.343609] Bluetooth: RFCOMM ver 1.11
[ 116.349936] hello: loading out-of-tree module taints kernel.
[ 116.350407] Hello World
pi@raspberrypi:~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules $ sudo rmmod hello
pi@raspberrypi:~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules $ dmesg | tail
[ 12.951840] Bluetooth: HCI UART protocol Broadcom registered
[ 13.276948] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[ 13.276961] Bluetooth: BNEP filters: protocol multicast
[ 13.276985] Bluetooth: BNEP socket layer initialized
[ 13.343554] Bluetooth: RFCOMM TTY layer initialized
[ 13.343586] Bluetooth: RFCOMM socket layer initialized
[ 13.343609] Bluetooth: RFCOMM ver 1.11
[ 116.349936] hello: loading out-of-tree module taints kernel.
[ 116.350407] Hello World
[ 150.700918] Goodbye World
pi@raspberrypi:~/EmbeddedSystems/Lab6_Kernel/Ubuntu/modules $
```



## 五、讨论与心得

本次实验的树莓派 Linux 内核源码编译部分主要参考官方文档：

<https://www.raspberrypi.org/documentation/linux/kernel/building.md>

在编译内核阶段也踩了几个坑：

一开始我使用 Mac 下载好源码，放到 Linux 虚拟机中编译，结果编译报错，说某些头文件找不到了。这是因为 Mac 是大小写不敏感的，而正好这个 Linux 源码中含有一些除大小写外都相同的文件名，在 git clone 的过程中 Mac 自动把它认为重复的文件删除了。

文件缺失可以手动下载相关文件补全，但是怕之后编译又出现其他问题，我还是重新在 Linux 里面下载了全部源码，至此编译总算是成功了，编译出的内核也能够在树莓派上运行。

第二个坑是在试图添加系统调用的时候，需要修改系统中断向量表，增加我自己的系统调用入口，但是在/arch/arm/kernel/目录下却找不到 calls.S 汇编文件了。上网一查才发现是因为比较新的 Linux 内核（我用的 4.18）提高了安全性，已经不会输出系统中断向量表了。

虽然也有办法找到中断向量表的内存地址，但是这样比较麻烦而且可能出现其他 bug，因此我不想冒险，决定换一个老版本的内核。

第三次失败是换了 3.18 的内核，再次编译，下载到板子上运行，结果树莓派启动失败，HDMI 显示器也花屏，原因未知。

或许是版本太老了哪里配置不对或者不支持，我于是又换了一个 4.9 的内核（拥有 calls.S 的版本中最新的），这次终于成功运行并顺利添加系统调用了。

添加内核模块时也踩了坑，直接在树莓派上编译报错，找不到 build 文件夹，然后我才想起是因为我的内核是交叉编译的，很多东西就没有安装到树莓派上。

于是还是在宿主机上交叉编译内核模块，至此内核模块添加成功。