

实验 09 编写简单应用调用驱动

9.1 本章导读

本期实验比较简单，就是写一个简单的应用程序调用前面写的驱动。

9.1.1 工具

9.1.1.1 硬件工具

- 1) iTOP4412 开发板
- 2) U 盘或者 TF 卡
- 3) PC 机
- 4) 串口

9.1.1.2 软件工具

- 1) 虚拟机 Vmware
- 2) Ubuntu12.04.2
- 3) 超级终端 (串口助手)
- 4) 实验配套源码文件夹 “invoke_hello”

9.1.2 预备课程

实验 08_生成设备节点

9.1.3 视频资源

本节配套视频为“视频 09-编写简单应用调用驱动”

9.2 学习目标

本章需要学习以下内容：

学会调用设备节点

9.3 实验操作

本期实验很简单，在前面 Linux 应用中就已经学习过设备节点的调用。

需要用到函数 `extern void printf(const char *format,...);` 定义在标准 C 语言头文件 `stdio.h` 中。

下面几个头文件在应用中一般一起调用。

头文件 `#include <sys/types.h>` 包含基本系统数据类型。系统的基本数据类型在 32 编译环境中保持为 32 位值,并会在 64 编译环境中增长为 64 位值。

头文件 `<sys/stat.h>` 包含系统调用文件的函数。可以调用普通文件、目录、管道、socket、字符、块的属性。

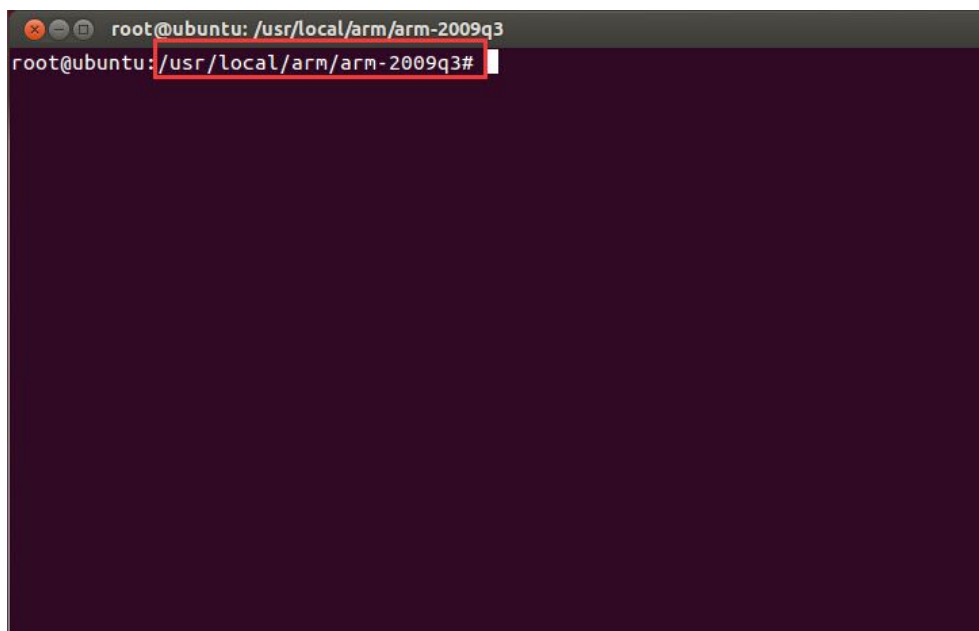
`<fcntl.h>` 定义了 `open` 函数

`<unistd.h>` 定义了 `close` 函数

`<sys/ioctl.h>` 定义了 `ioctl` 函数

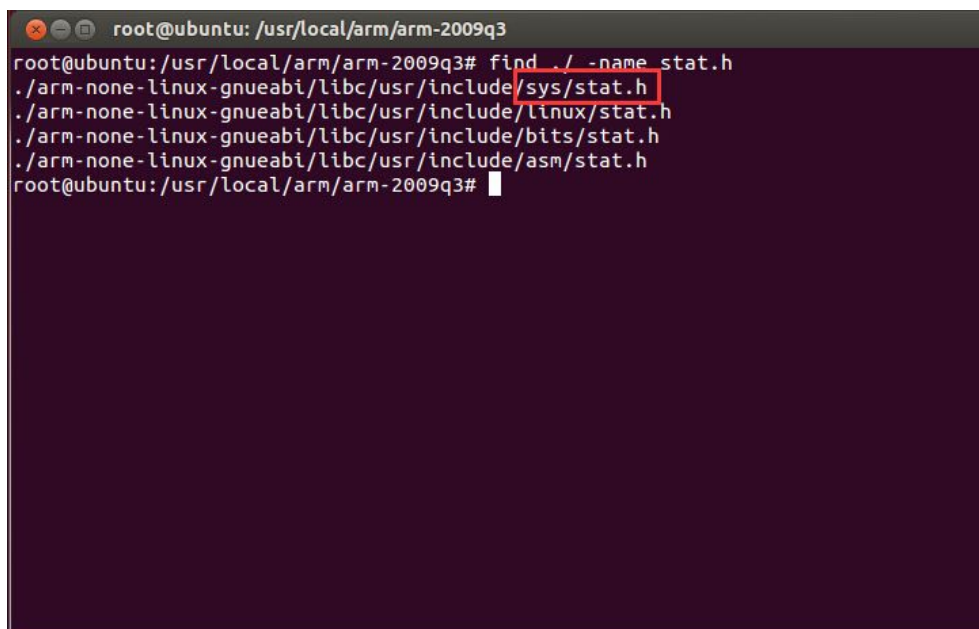
另外提醒一下，这些头文件是和编译器在一起。

这里使用，如下图所示，进入目录 “/usr/local/arm/arm-2009q3” 。

A terminal window with a dark background. The title bar shows 'root@ubuntu: /usr/local/arm/arm-2009q3'. The command prompt is 'root@ubuntu: /usr/local/arm/arm-2009q3#'. The path '/usr/local/arm/arm-2009q3#' is highlighted with a red box.

```
root@ubuntu: /usr/local/arm/arm-2009q3
root@ubuntu: /usr/local/arm/arm-2009q3#
```

使用查找命令 “find ./ -name stat.h” ,如下图所示，使用的头文件是目录 “/arm-none-linux-gnueabi/libc/usr/include/sys/stat.h” 中的<sys/stat.h>。

A terminal window with a dark background. The title bar shows 'root@ubuntu: /usr/local/arm/arm-2009q3'. The command prompt is 'root@ubuntu: /usr/local/arm/arm-2009q3#'. The command 'find ./ -name stat.h' has been entered. The output shows four paths, with the first one, './arm-none-linux-gnueabi/libc/usr/include/sys/stat.h', highlighted by a red box.

```
root@ubuntu: /usr/local/arm/arm-2009q3# find ./ -name stat.h
./arm-none-linux-gnueabi/libc/usr/include/sys/stat.h
./arm-none-linux-gnueabi/libc/usr/include/linux/stat.h
./arm-none-linux-gnueabi/libc/usr/include/bits/stat.h
./arm-none-linux-gnueabi/libc/usr/include/asm/stat.h
root@ubuntu: /usr/local/arm/arm-2009q3#
```

其它几个头文件可以采用类似的方法查找，这里给大家提醒这一点，因为有时候拿到源码之后，可能编译器版本和源码不完全对应，这个时候就有可能需要修改和处理一下头文件。不过这种问题一般都可以通过网络查找错误提示的方法一个一个解决。

如下图所示，是一个简单的调用程序。

```
#include <stdio.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>

main() {
    int fd;
    char *hello_node = "/dev/hello_ctl123";

    /*O_RDONLY只读打开,O_NDELAY非阻塞方式*/
    if((fd = open(hello_node,O_RDONLY|O_NDELAY))<0) {
        printf("APP open %s failed",hello_node);
    }
    else{
        printf("APP open %s success",hello_node);
        ioctl(fd,1,6);
    }

    close(fd);
}
```

新建“invoke_hello”文件夹，将上图的中的文件拷贝进入，进入新建的“invoke_hello”目录，使用编译命令

“arm-none-linux-gnueabi-gcc -o invoke_hello invoke_hello.c -static”

编译，如下图所示。

```
root@ubuntu: /home/topeet/invoke_hello
root@ubuntu:~# cd /home/topeet/
root@ubuntu:/home/topeet# mkdir invoke_hello
root@ubuntu:/home/topeet# cd invoke_hello/
root@ubuntu:/home/topeet/invoke_hello# ls
invoke_hello.c
root@ubuntu:/home/topeet/invoke_hello# arm-none-linux-gnueabi-gcc -o invoke_hello invoke_hello.c -static
root@ubuntu:/home/topeet/invoke_hello# ls
invoke_hello invoke_hello.c
root@ubuntu:/home/topeet/invoke_hello#
```

将“invoke_hello”拷贝到 U 盘 ,启动开发板 ,加载前一期的“devicenode_linux_module”驱动 ,如下图所示 ,使用 invoke_hello 调用设备节点 “/dev/hello_ctl123” 。

先使用命令 “mount /dev/sda1 /mnt/udisk/” 加载 U 盘 ;

使用命令 “insmod /mnt/udisk/devicenode_linux_module.ko” 加载驱动 ;

使用命令 “./mnt/udisk/invoke_hello” ,运行 invoke_hello。

```
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]# insmod /mnt/udisk/devicenode_linux_module.ko
[ 32.363393] HELLO WORLD enter!
[ 32.365480] initialized
[ 32.380682] DriverState is 0
[root@iTOP-4412]# [ 33.206858] CPU3: shutdown
[root@iTOP-4412]# ./mnt/udisk/invoke_hello
int_helloworld invoke_char_gpios invoke_hello
invoke_char_driver invoke_gpios invoke_leds
[root@iTOP-4412]# ./mnt/udisk/invoke_hello
[ 49.050442] hello open
[ 49.056632] cmd is 1,arg is 6
[ 49.058159] hello release
APP open /dev/hello_ctl123 success[root@iTOP-4412]#
```

如上图所示 ,运行 “invoke_hello” 之后 ,会打印以下内容”

hello open

cmd is 1,arg is 6

hello release

如下图所示，设备节点 open、close、ioctl 分别对应打印信息

```
printk(KERN_EMERG "hello open\n");
```

```
printk(KERN_EMERG "hello release\n");
```

```
printk("cmd is %d,arg is %d\n",cmd,arg);
```

ioctl 会打印第二个和第三个参数。

```
static long hello_ioctl( struct file *files, unsigned int cmd, unsigned long arg){  
  
    printk("cmd is %d,arg is %d\n",cmd,arg);  
    return 0;  
}  
  
static int hello_release(struct inode *inode, struct file *file){  
    printk(KERN_EMERG "hello release\n");  
    return 0;  
}  
  
static int hello_open(struct inode *inode, struct file *file){  
    printk(KERN_EMERG "hello open\n");  
    return 0;  
}
```

通过前面的分析，可以看到上层应用对设备节点 open、close、ioctl 分别对应驱动层的 open、release、unlocked_ioctl。