https://wy-wulian.taobao.com/

第 15 章　驱动开发之字符设备驱动程序之概念介绍

本章目标

- 对 Linux 内核、驱动有初步的认识

15.1　应用程序、库、内核、驱动程序的关系

从上到下，一个软件系统可以分为：应用程序、库、操作系统(内核)、驱动程序。开发人员可以专注于自己熟悉的部分，对于相邻层，只需要了解它的接口，无需关注它的实现细节。

以点亮一个 LED 为例，这 4 层软件的协作关系如下，如图 1 所示。

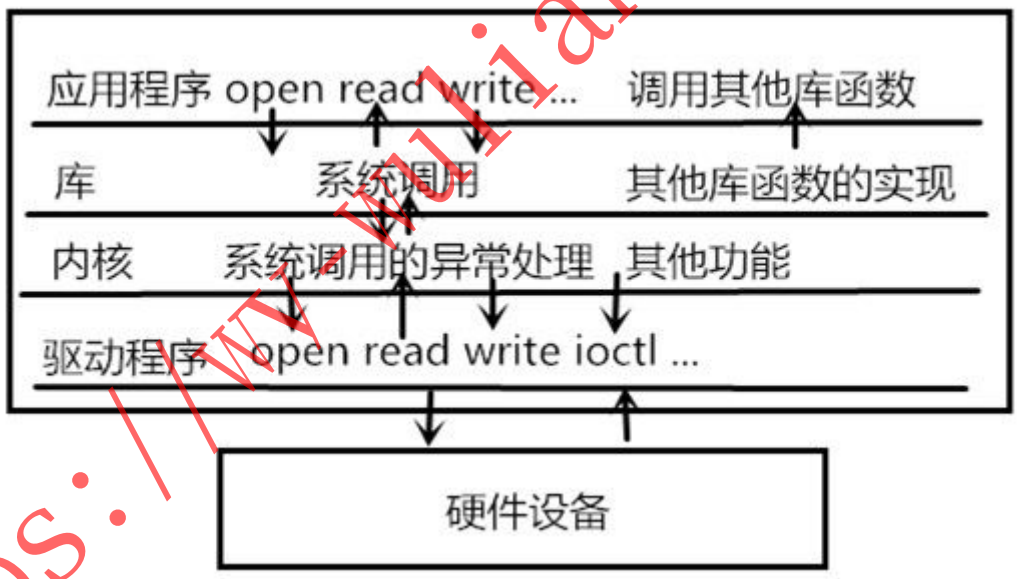

图 1

1)．应用程序使用库提供的 open 函数打开代表 LED 的设备文件。

2)．库根据 open 函数传入的参数执行“SWI”指令，这条指令会引起 CPU 异常，进入内核。

3)．内核的异常处理函数根据这些参数找到相应的驱动程序，返回一个文件句柄给库，进而返回给应用程序。

https://www.wulian.taobao.com/

4). 应用程序得到文件句柄后，使用库提供的 write 或 ioclt 函数发出控制命令。

5). 库根据 write 和 ioclt 函数传人的参数执行“swi”指令，这条指令会引起 CPU 异常，进入内核。

6). 内核的异常处理函数根据这些参数调用驱动程序的相关函数，点亮 LED。

库(比如 glibc)给应用程序提供的 open、read、write、ioctl、mmap 等接口函数被称为系统调用，它们都是设置好相关寄存器后，执行某条指令引发异常进入内核。除系统调用接口外，库还提供其他函数，比如字符串处理函数(strcpy、strcmp 等)、输入/输出函数(scanf、printf 等)、数学库，还有应用程序的启动代码等。

在异常处理函数中，内核会根据传入的参数执行各种操作，比如根据设备文件名找到对应的驱动程序，调用驱动程序的相关函数等。

一般来说，当应用程序调用 open、read、write、ioctl、mmap 等函数后，将会使用驱动程序中的 open、read、write、ioctl、mmap 函数来执行相关操作，比如初始化、读、写等。

实际上，内核和驱动程序之间并没有界线，因为驱动程序最终是要编进内核去的：通过静态链接和动态加载。

从上面操作 LED 的过程可以知道，与应用程序不同，驱动程序从不主动运行，它是被动的：根据应用程序的要求进行初始化，根据应用程序的要求进行读写。驱动程序加载进内核时，只是告诉内核“我在这里，我能做这些工作”，至于这些“工作”何时开始，取决于应用程序。当然，这不是绝对的，比如用户完全可以写一个系统时钟触发的驱动程序，让它自动点亮 LED。

在 Linux 系统中，应用程序运行于“用户空间”，拥有 MMU 的系统能够限制应用程序的权限(比如将它限制于某个内存块中)，这可以避免应用程序的错误使整个系统崩溃。而驱动程序运行于“内核空间”，它是系统“信任”的一部分，驱动程序的错误有可能导致整个系统崩溃。

15.2 Linux 驱动程序分类

Linux 的外设可以分为 3 类：自费设备、块设备和网络接口。

字符设备是能够像字节流(比如文件)一样被访问的设备，就是说对它的读写是以字节为单位的。比如串口在进行收发数据时就是一个字节一个字节的进行的，我们可以在驱动程序内部使用缓冲区来存放数据以提高效率，但是串口本身对这并没有要求。字符设备的驱动程序中实现了 open、close、read、write 等系统调用，应用程序可以通过设备文件(比如/dev/ttySAC0 等)来访问字符设备。

块设备上的数据以块的形式存放，比如 NAND Flash 上的数据就是以页为单位存放的。块设备驱动程序向用户层提供的接口与字符设备一样，应用程序也可以通过相应的设备文件(比如/dev/mtdblock0、/dev/hda1 等)来调用 open、close、read、write 等系统调用，与块设备传送任意字节的数据。对用户而言，字符设备和块设备的访问方式没有差别。块设备驱动程序的特别之处如下。

1). 操作硬件的接口实现方式不一样。

https://wy-wulian.taobao.com/

块设备驱动程序先将用户发来的数据组织成块，再写入设备；或从设备中读出若干块数据，再从中挑出用户需要的。

2). 数据块上的数据可以有一定的格式。

通常在块设备中按照一定的格式存放数据，不同的文件系统类型就是用来定义这些格式的。内核中，文件系统的层次位于块设备驱动程序上面，这意味着块设备驱动程序除了向用户层提供与字符设备一样的接口外，还要向内核其他部件提供一些接口，这些接口用户是看不到的。这些接口使得可以在块设备上存放文件系统，挂载块设备。

网络接口同时具有字符设备、块设备的部分特点，无法将它归入这两类中：如果说它是字符设备，他的输入/输出却是有结构的、成块的(报文、包、帧)；如果说它是块设备，它的“块”又不是固定大小的，大到数百甚至数千字节，小到几字节。UNIX 式的操作系统访问网络接口的方法是给它们分配一个惟一的名字(比如 eth0)，但这个名字在文件系统中(比如/dev 目录下)不存在对应的节点项。应用程序、内核和网络驱动程序间的通信完全不同于字符设备、块设备，库、内核提供了一套和数据包传输相关的函数，而不是 open、read、write 等。

15.3 Linux 驱动程序开发步骤

Linux 内核就是由各种驱动组成的，内核源码中有大约 85%是各种驱动程序的代码。内核中驱动程序种类齐全，可以在同类驱动的基础上进行修改以符合具体单板。

编写驱动程序的难点并不是硬件的具体操作，而是弄清楚现有驱动程序的框架，在这个框架中加入这个硬件。比如，x86 架构的内核对 IDE 硬盘的支持非常完善：首先通过 BIOS 得到硬盘的信息，或者使用默认 I/O 地址去枚举硬盘，然后识别分区、挂载文件系统。对于其他架构的内核，只是要指定了硬盘的访问地址和中断号，后面的枚举、识别和挂接的过程完全是一样的。也许修改的代码不超过 10 行，花费精力的地方在于：了解硬盘驱动的框架，找到修改的位置。

编写驱动程序还有很多需要注意的地方，比如：驱动程序可能同时被多个进程使用，这需要考虑并发的问题；尽可能发挥硬件的作用以提高性能。比如在硬盘驱动程序中既可以使用 DMA 也可以不用，使用 DMA 时程序比较复杂，但是可以提高效率；处理硬件的各种异常情况(即使效率低)，否则出错时可能导致整个系统崩溃。

一般来说，编写一个 Linux 设备驱动程序的大致流程如下。

1). 查看原理图、数据手册，了解设备的操作方法。

2). 在内核中找到相近的驱动程序，以它为模板进行开发，有时候需要从零开始。

3). 实现驱动程序的初始化：比如向内核注册这个驱动程序，这样应用程序传入文件名时，内核才能找到相应的驱动程序。

4). 设计所要实现的操作，比如 open、close、read、write 等函数。

5). 实现中断服务(中断并不是每个设备驱动所必须的)。

6). 编译该驱动程序到内核中，或者用 insmod 命令加载。

7). 测试驱动程序。

https://wy-wulian.taobao.com/

15.4 驱动程序的加载和卸载

可以将驱动程序静态编译进内核中，也可以将它作为模块在使用时再加载。在配置内核时，如果某个配置选项被设为 m，就表示它将会被编译成一个模块。在 2.6 的内核中，模块的扩展名为.ko，可以使用 insmod 命令加载，使用 rmmod 命令卸载，使用 lsmod 命令查看内核中已经加载了哪些模块。

当使用 insmod 加载模块时，模块的初始化函数被调用，它用来向内核注册驱动程序；当使用 rmmod 卸载模块时，模块的清除函数被调用。在驱动代码中，这两个函数要么取固定的名字：init_module 和 cleanup_module，要么使用以下两行来标记它们(假设初始化函数、清除函数为 my_init 和 my_cleanup)。

```
moudle_init(my_init);
module_exit(my_cleanup);
```

注意：

1). 该教程为我司(https://wy-wulian.taobao.com/)原创教程，版权所有；

2). 该教程会不断更新、不断深入，详情请咨询我司客服；

3). 针对该教程，我们还有 QQ 群和论坛，专门负责技术答疑，详情请咨询我司客服。

https://wy-wulian.taobao.com/