

Android 开发之 ---- 底层驱动开发(一)

说到 android 驱动是离不开 Linux 驱动的。Android 内核采用的是 Linux2.6 内核（最近 Linux 3.3 已经包含了一些 Android 代码）。但 Android 并没有完全照搬 Linux 系统内核，除了对 Linux 进行部分修正，还增加了不少内容。**android 驱动主要分两种类型 Android 专用驱动 和 Android 使用的设备驱动（linux）。**

Android 专有驱动程序：

1) Android Ashmem **匿名共享内存**：为用户空间程序提供分配内存的机制，为进程间提供大块共享内存，同时为内核提供回收和管理这个内存。

2) Android Logger 轻量级的 LOG（日志）驱动：

3) Android Binder 基于 OpenBinder 框架的一个驱动：

4) Android Power Management 电源管理模块：

5) Low Memory Killer 低内存管理器：

6) Android PMEM 物理内存驱动：

7) USB Gadget USB 驱动（**基于 gaaget 框架**）：

8) Ram Console 用于调试写入日志信息的设备：

9) Time Device 定时控制设备：

10) Android Alarm 硬件时钟：

Android 上的设备驱动（linux）：

1) **Framebuffer** 显示驱动：

2) **Event** 输入设备驱动：

3) **ALSA** 音频驱动：

4) **OSS** 音频驱动：

5) **v412** 摄像头：视频驱动：

6) **MTD** 驱动：

7) **蓝牙**驱动：

8) **WLAN** 设备驱动：

Android 专有驱动程序

1.Android Ashmem

为用户空间程序提供分配内存的机制，为进程间提供大块共享内存，同时为内核提供回收和管理这个内存。

设备节点：**/dev/ashmem** .主设备号 **10**。

源码位置：**include/linux/ashmem.h Kernel /mm/ashmem.c**

相比于 **malloc** 和 **anonymous/named mmap** 等传统的内存分配机制，其优势是通过内核驱动提供了辅助内核的内存回收算法机制（**pin/unoin**）

2.Android Logger

无论是底层的源代码还上层的应用，我们都可以使用 `logger` 这个日志设备看、来进行调试。

设备节点: `/dev/log/main` `/dev/log/event` `/dev/log/radio`

源码位置: `include/linux/logger.h` `include/linux/logger.c`

3.Android Binder

`IPC Binder` 一种进程间通信机制。他的进程能够为其它进程提供服务--通过标准的 `Linux` 系统调用 `API`。

设备节点 : `/dev/binder`

源码位置: `Kernel/include/linux/binder.h` `Kernel/drivers/misc/binder.c`

4.Android Power Management

一个基于标准 `linux` 电源管理的轻量级 `Android` 电源管理系统，

源码位置: `drivers/android/power.c` `kernel/power/`

5.Low Memory Killer

它在用户空间中指定了一组内存临界值，当其中某个值与进程描述中的 `oom_adj` 值在同一范围时，该进程将被 `Kill` 掉（在 `parameters/adj` 中指定 `oom_adj` 的最小值）。它与标准的 `Linux OOM` 机制类似，只是实现方法不同

源码位置: `drivers/misc/lowmemorykiller.c`

6.Android PMEM

`PMEM` 主要作用就是向用户空间提供连续的物理内存区域。

1.让 `GPU` 或 `VPU` 缓冲区共享 `CPU` 核心。2.用于 `Android service` 堆。

源码位置: `include/linux/android_pmem.h` `drivers/android/pmem.c`

7.USB Gadget

基于标准 `Linux USB gaeget` 驱动框架的设备驱动。

源码位置: `drivers/usb/gadet/`

8.Ram Console

为了提供调试功能，`android` 允许将调试日志信息写入这个设备，它是基于 `RAM` 的 `buffer`。

源码位置: `drivers/staging/android/ram_console.c`

9.Time Device

定时控制,提供了对设备进行定时控制的功能。

源码位置: `drivers/staging/android/timed_output.c(timed_gpio.c)`

10.Android Alarm

提供一个定时器，用于把设备从睡眠状态唤醒，同时它还提供了一个即使在设备睡眠时也会运行的时钟基准。

设备节点：`/dev/alarm`

源码位置：`drivers/ttc/alarm.c`

Android 设备驱动

1. Framebuffer 帧缓存设备

Framebuffer 驱动在 Linux 中是标准的显示设备的驱动。对于 PC 系统，它是显卡的驱动；对于嵌入式 SOC 处理器系统，它是 LCD 控制器或者其他显示控制器的驱动。它是一个字符设备，在文件系统中设备节点通常是 `/dev/fbx`。每个系统可以有多个显示设备，依次用 `/dev/fb0`，`/dev/fb1` 等来表示。在

Android 系统中主设备号为 29，次设备号递增生成。

Android 对 Framebuffer 驱动的使用方式是标准的，在 `/dev/graphics` 中的 Framebuffer 设备节点由 `init` 进程自动创建，被 `libui` 库调用。Android 的 GUI 系统中，通过调用 Framebuffer 驱动的标准接口，实现显示设备的抽象。

Framebuffer 的结构框架和实现：

[linux LCD 驱动（二）--FrameBuffer](#)

[Linux LCD 驱动\(四\) --驱动的实现](#)

2.Event 输入设备驱动

Input 驱动程序是 Linux 输入设备的驱动程序，分为游戏杆 (joystick)、鼠标 (mouse 和 mice) 和事件设备 (Event queue) 3 种驱动程序。其中事件驱动程序是目前通用的程序、可支持键盘、鼠标、触摸屏等多种输入设备。Input 驱动程序的主设备号是 13，每一种 Input 设备从设备号占用 5 位，3 种从设备号分配是：游戏杆 0 ~ 61；Mouse 鼠标 33 ~ 62；Mice 鼠标 63；事件设备 64 ~ 95，各个具体的设备在 `misc`、`touchscreen`、`keyboard` 等目录中。

Event 设备在用户空间使用 `read`、`ioctl`、`poll` 等文件系统的接口操作，`read` 用于读取输入信息，`ioctl` 用于获取和设置信息，`poll` 用于用户空间的阻塞，当内核有按键等中断时，通过在中断中唤醒内核的 `poll` 实现。

Event 输入驱动的架构和实现：

[Linux 设备驱动之——input 子系统](#)

3.ALSA 音频驱动

高级 Linux 声音体系 ALSA (Advanced Linux Sound Architecture) 是为音频系统提供驱动的 Linux 内核组件，以替代原先的开发声音系统 OSS。它是一个完全开放源代码的音频驱动程序集，除了像 OSS

那样提供一组内核驱动程序模块之外，ALSA 还专门为简化应用程序的编写提供相应的函数库，与 OSS 提供的基于 `ioctl` 等原始编程接口相比，ALSA 函数库使用起来要更加方便一些

利用该函数库，开发人员可以方便、快捷地开发出自己的应用程序，细节则留给函数库进行内部处理。所以虽然 ALSA 也提供了类似于 OSS 的系统接口，但建议应用程序开发者使用音频函数库，而不是直接调用驱动函数。

ALSA 驱动的主设备号为 116，次设备号由各个设备单独定义，主要的设备节点如下：

`/dev/snd/controlC#` —— 主控制；

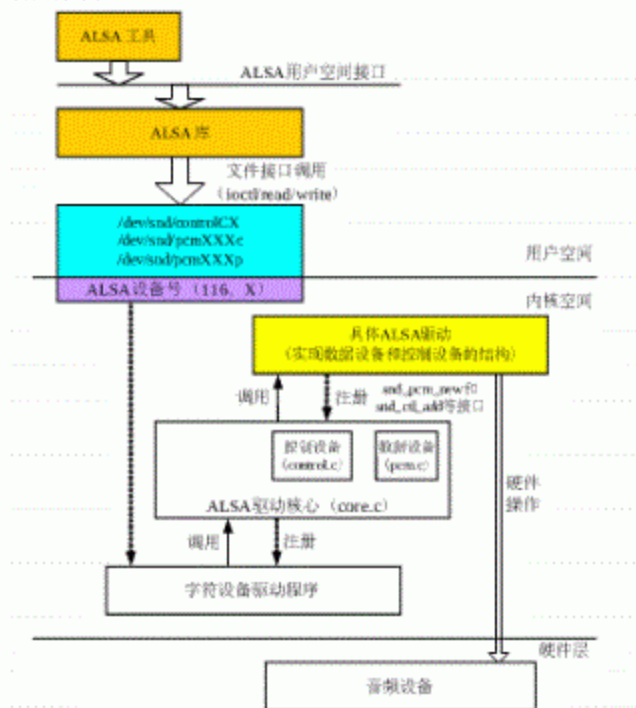
`/dev/snd/pcmXXXc` —— PCM 数据通道；

`/dev/snd/seq` —— 顺序器；

`/dev/snd/timer` —— 定义器。

在用户空间中，ALSA 驱动通常配合 `alsa` 库使用，库通过 `ioctl` 等接口调用 ALSA 驱动程序的设备节点。对于 ALSA 驱动的调用，调用的是用户空间的 ALSA 库的接口，而不是直接调 ALSA 驱动程序。

ALSA 音频驱动的架构



ALSA 驱动程序的主要头文件是 `include/sound/sound.h`，驱动核心数据结构和具体驱动的注册函数是 `include/sound/core.h`，驱动程序的核心实现是 `Sound/core/sound.c` 文件。

ALSA 驱动程序使用下面的函数注册控制和设备：

```
int snd_pcm_new(struct snd_card * card, char * id, int device, int playback_count, int capture_count, struct snd_pcm ** rpcm) ;
```

```
int snd_ctl_add(struct snd_card * card, struct snd_kcontrol * kcontrol1) ;
```

ALSA 音频驱动在内核进行 menuconfig 配置时，配置选项为 “Device Drivers” > “Sound card support” —> “Advanced Linux Sound Architecture”。子选项包含了 Generic sound devices(通用声音设备)、ARM 体系结构支持，以及兼容 OSS 的几个选项。ALSA 音频驱动配置对应的文件是 sound / core / Kconfig。

Android 没有直接使用 ALSA 驱动，可以基于 A-LSA 驱动和 ALSA 库实现 Android Audio 的硬件抽象层：ALSA 库调用内核的 ALSA 驱动，Audio 的硬件抽象层调用 ALSA 库。

4.OSS 音频驱动

OSS (Open Sound System 开放声音系统) 是 linux 上最早出现的声卡驱动。OSS 由一套完整的内核驱动程序模块组成，可以为绝大多数声卡提供统一的编程接口。

OSS 是字符设备，主设备号 14，主要包括下面几种设备文件：

1) /dev/sndstat

它是声卡驱动程序提供的简单接口，它通常是一个只读文件，作用也只限于汇报声卡的当前状态。(用于检测声卡)

2) /dev/dsp

用于数字采样和数字录音的设备文件。对于音频编程很重要。实现模拟信号和数字信号的转换。

3) /dev/audio

类似于/dev/dsp，使用的是 mu-law 编码方式。

4) /dev/mixer

用于多个信号组合或者叠加在一起，对于不同的声卡来说，其混音器的作用可能各不相同。

5) /dev/sequencer

这个设备用来对声卡内建的波表合成器进行操作，或者对 MIDI 总线上的乐器进行控制。

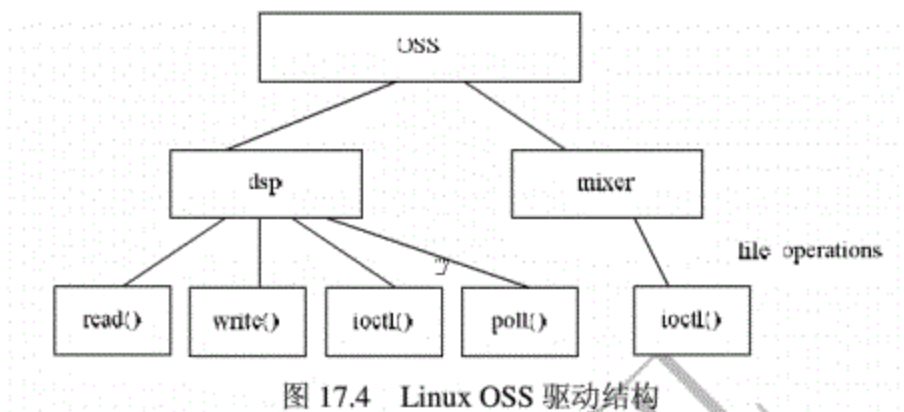
OSS 驱动所涉及的文件主要包括：

kernel/include/linux/soundcard.h

kernel/include/linux/sound.h 定义 OSS 驱动的次设备号和注册函数

kernel/sound_core.c OSS 核心实现部分

OSS 驱动架构图：



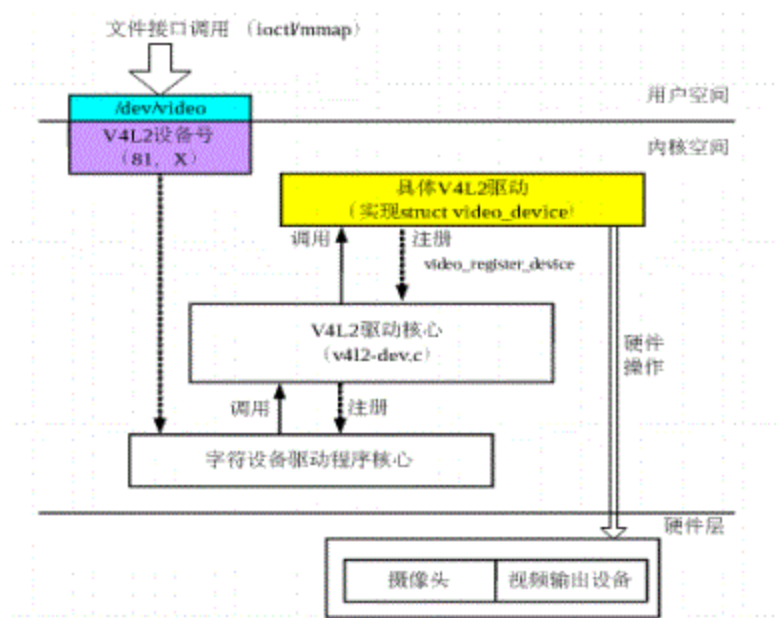
5.V4L2 视频驱动

V4L2 是 V4L 的升级版，为 linux 下视频设备程序提供了一套接口规范。包括一套数据结构和底层 V4L2 驱动接口。V4L2 提供了很多访问接口，你可以根据具体需要选择操作方法。需要注意的是，很少有驱动完全实现了所有的接口功能。所以在使用时需要参考驱动源码，或仔细阅读驱动提供者的使用说明。

V4L2 的主设备号 是 81，次设备号：0~255，这些次设备号里也有好几种设备（视频设备、Radio 设备、Teletext、VBI）。

V4L2 的设备节点：/dev/videoX, /dev/vbiX and /dev/radioX

V4L2 框架图：



Android 开发之 ---- 底层驱动开发(二)

MTD 驱动

Flash 驱动通常使用 MTD (memory technology device)，内存技术设备。

MTD 的字符设备：/dev/mtdX 主设备号 90.

MTD 的块设备：/dev/block/mtdblockX 主设备号 13.

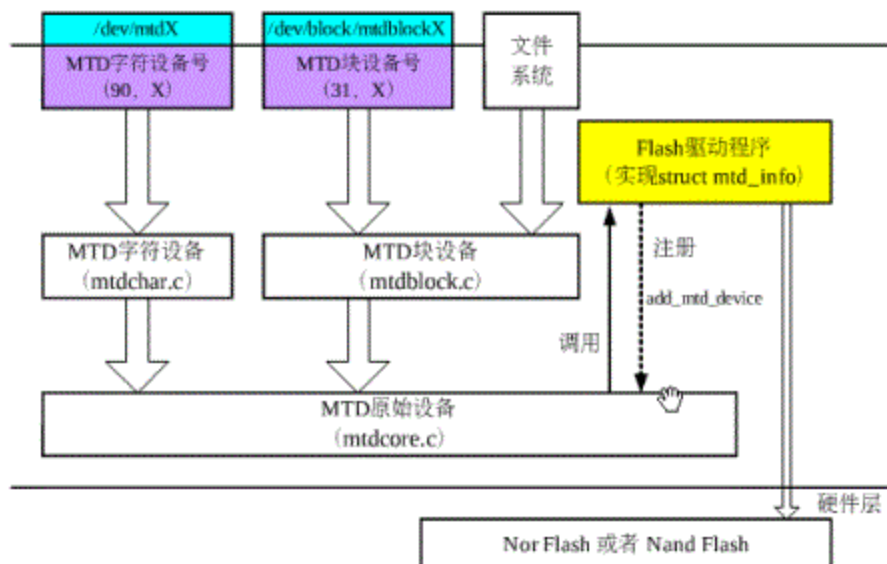
MTD 驱动源码

drivers/mtd/mtdcore.c: MTD 核心, 定义 MTD 原始设备

drivers/mtd/mtdchar.c: MTD 字符设备

drivers/mtd/mtdblock.c: MTD 块设备

MTD 结构图



MTD 驱动程序是 Linux 下专门为嵌入式环境开发的新一类驱动程序。Linux 下的 MTD 驱动程序接口被划分为用户模块和硬件模块：

用户模块 提供从用户空间直接使用的接口：原始字符访问、原始块访问、FTL（Flash Transition Layer）和 JFS（Journaled File System）。

硬件模块 提供内存设备的物理访问，但不直接使用它们，而是通过上述的用户模块来访问。这些模块提供了闪存上读、写和擦除等操作的实现。

蓝牙驱动

在 Linux 中，蓝牙设备驱动是网络设备，使用网络接口。

Android 的蓝牙协议栈使用 BlueZ 实现来对 GAP, SDP 以及 RFCOMM 等应用规范的支持,并获得了 SIG 认证。由于 Bluez 使用 GPL 授权, 所以 Android 框架通过 D-BUS IPC 来与 bluez 的用户空间代码交互以避免使用未经授权的代码。

蓝牙协议部分头文件: include/net/bluetooth/hci_core.h include/net/bluetooth/bluetooth.h

蓝牙协议源代码文件: net/bluetooth/*

蓝牙驱动程序部分的文件: drivers/bluetooth/*

蓝牙的驱动程序一般都通过标准的 HCI 控制实现。但根据硬件接口和初始化流程的不同,又存在一些差别。这类初始化动作一般是一些晶振频率,波特率等基础设置。比如 CSR 的芯片一般通过 BCSP 协议完成最初的初始化配置,再激活标准 HCI 控制流程。对 Linux 来说,一旦 bluez 可以使用 HCI 与芯片建立起通信(一般是 hciattach + hciconfig),便可以利用其上的标准协议(SCO, L2CAP 等),与蓝牙通信,使其正常工作了。

WLAN 设备驱动 (Wi-Fi) (比较复杂我后面会专门写个 wifi 分析)

在 linux 中, Wlan 设备属于网络设备, 采用网络接口。

Wlan 在用户空间采用标准的 socket 接口进行控制。

WiFi 协议部分头文件: include/net/wireless.h

WiFi 协议部分源文件: net/wireless/*

WiFi 驱动程序部分: drivers/net/wireless/*

wifi 模块结构图:

