**内核固件接口**

1. int request\_firmware(const struct firmware \*\*firmware\_p, const char \*name, struct device \*device) //同步请求

作用：

请求用户空间为内核 **定位** 固件镜像文件 以及 将定位的镜像文件**写入到属性文件** /sys/class/firmware/devicename/data

详细过程：

1). 创建属性文件,如loading，data，timeout

2). 当调用request\_firmware()的时候，在该接口中调用了产生热插拔事件的接口，kobject\_hotplug(&class\_dev->kobj, KOBJ\_ADD);

3). 调用wait\_for\_completion(&fw\_priv->completion);等待固件镜像文件加载到data属性完成后返回

4). 在request\_firmware()成功调用返回后，镜像文件的内容通过firmware\_p返回

5). 将正确的固件镜像加载到硬件如：

request\_firmware(&firmware, "BCM2033-MD.hex", &udev->dev);

memcpy(data->buffer, firmware->data, firmware->size);

usb\_fill\_bulk\_urb(data->urb, udev, usb\_sndbulkpipe(udev, BCM203X\_OUT\_EP), data->buffer, firmware->size, bcm203x\_complete, data);

release\_firmware(firmware);

6). 在该接口中，需要与用户空间交互，在返回前它需要进行休眠。

1. int request\_firmware\_nowait(struct module \*module, const char \*name, struct device \*device, void \*context, void (\*cont)(const struct firmware \*fw, void \*context))

说明：

当用户空间在加载固件程序完全之前，该函数不会休眠

该函数其实是调用了kernel\_thread(request\_firmware\_work\_func, fw\_work, CLONE\_FS | CLONE\_FILES);创建了一个内核线程，创建完后就返回了，而对于线程执行函数request\_firmware\_work\_func()，其实是通过调用request\_firmware()来实现的，它的休眠是通过在线程执行函数来实现的，而并未在request\_firmware\_nowait()中进行休眠，创建完线程后就返回了。

1. void release\_firmware(const struct firmware \*fw)

说明：

在把固件发送到设备后，使用该接口释放内核中的结构

firmware结构:

struct firmware {

size\_t size;

u8 \*data;

};

**固件子系统工作原理**

说明：

固件子系统加载固件程序 基于 **sysfs文件系统** 和 **热插拔机制**。

**sysfs文件系统起到的作用**：

\*当内核调用request\_firmware()的时候会以设备的名在sys/class/firmware下创建一个目录。

\*在该目录下会创建三个属性文件------loading，data，device

\*loading属性：就相当于一个flag用于标识固件下载的状态(下载中，下载完成，下载失败)

\*data属性：二进制属性文件，用户空间会把固件程序写入到该属性文件

\*device属性：一个指向/sys/devices下相应入口的符号链接

**热插拔机制起到的作用**：

\*当sysfs入口被创建的时候，会产生热插拔事件。

\*传递环境变量FIRMWARE给热插拔处理程序，该变量的值为固件的名称

\*热插拔处理程序定位(如在/etc/firmware下存放固件程序)固件文件，使用提供的属性文件data，将固件拷贝到内核

\*若处理程序没有发现固件文件，会将loading设置为-1。

**设置固件请求超时**：

\*若在timeout之内不能为固件请求提供服务，内核将放弃并向驱动程序返回错误状态。

\*timeout属性：在/sys/class/firmware/timeout