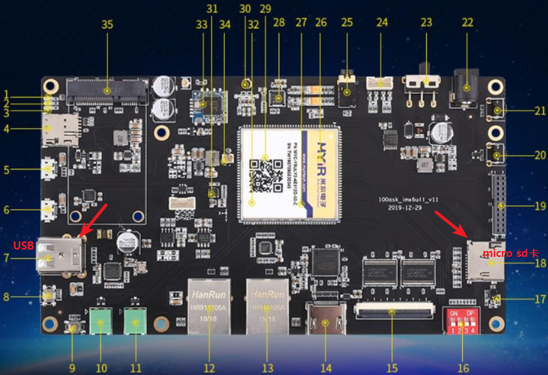
# 第四篇. 应用开发

## 15. 存储设备

### 15.1 SD/TF卡，U盘使用步骤

#### 15.1.1 SD/TF卡，U盘的硬件接口



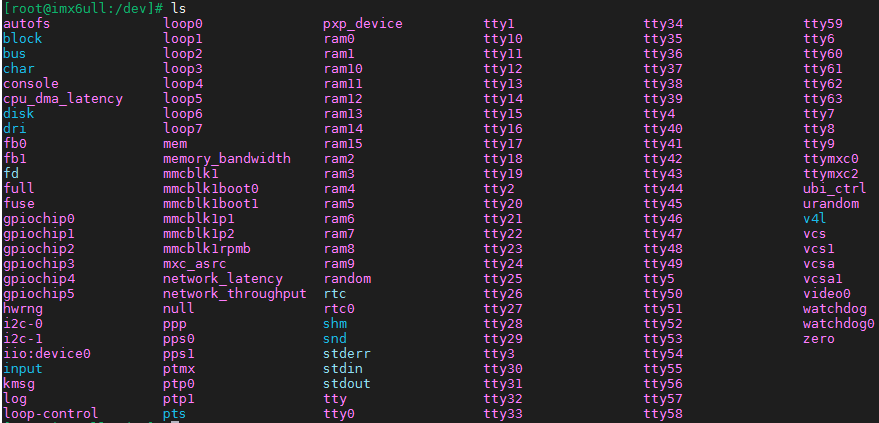
如上图的100ask6ull开发板，⑦为USB口，U盘从这里插入；⑱为mico sd卡槽，micro sd卡可以直接从这里插入。

注：micro sd卡也叫tf卡，它是sd卡类的一种，还有sd卡，mini sd卡等，我们简称这一类都叫sd卡。

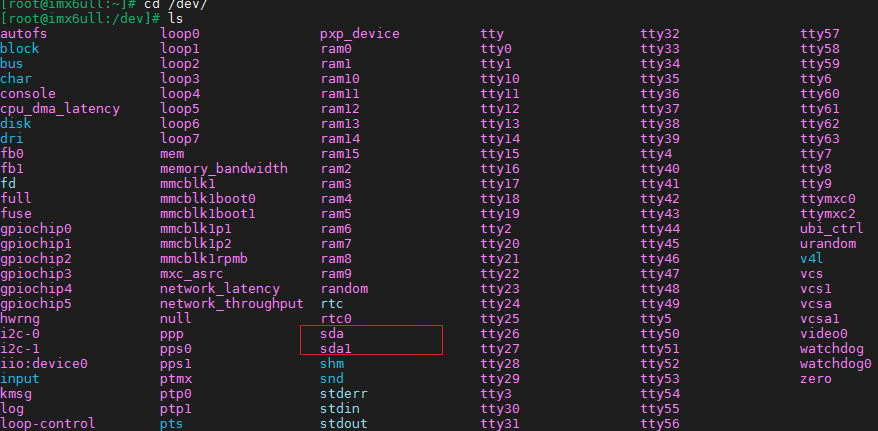
#### 15.1.2 确定设备点

①U盘确定设备点

下图是未插入U盘前/dev/目录下的内容

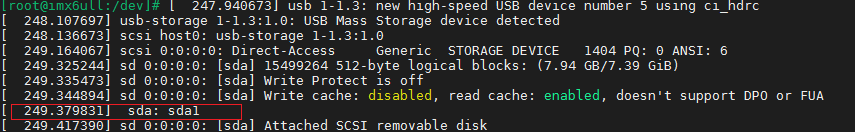


当我们插入U盘后/dev/目录下的内容为下图



由这两幅图，可以清楚看到插入U盘后，/dev/目录下多了sda，sda1，很显然这两个东西就是和我们的U盘有关的。

当我们插入U盘linux系统一般都会提示该设备节点是哪个，下图就是当我们插入U盘后我们串口终端接收的信息，上面提示了有U盘插入，同时设备节点是sda



        sd表示这是个scsi，sata硬盘设备。

        a：表示这是第一块硬盘，

1：这是这块硬盘的第一个分区，同时也是主分区。

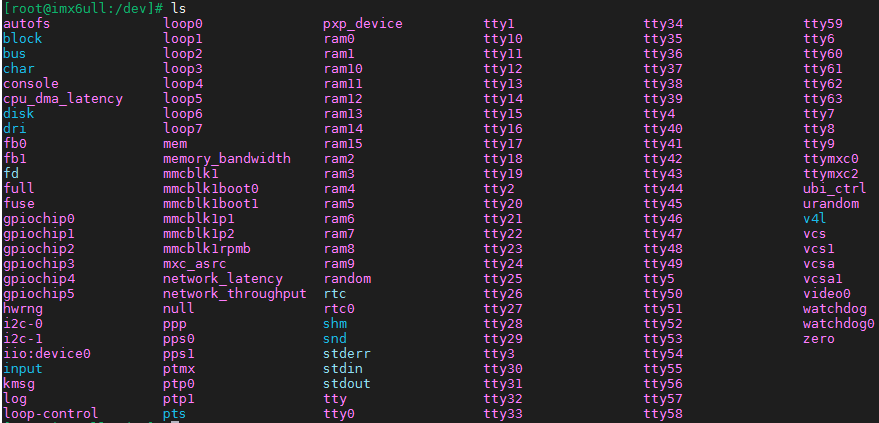
根据以上内容，我们应该就能看出sda，sda1表示什么意思了。

sda表示整个U盘存储设备

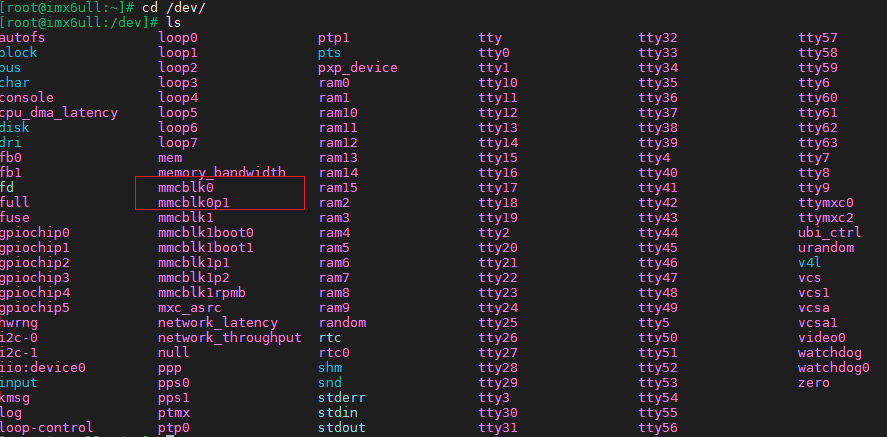
sda1表示是sda的第一个分区。

②SD卡确定设备点

下图是未插入SD卡前/dev/目录下的内容

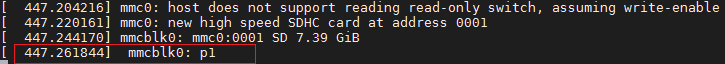


当我们插入SD卡后/dev/目录下的内容为下图



由这两幅图，可以清楚看到插入SD卡后，/dev/目录下多了mmcblk0，mmcblk0p1，很显然这两个东西就是和我们的SD卡有关的。

当我们插入sd卡，linux系统一般都会提示该设备节点是哪个，下图就是当我们插入sd卡后我们串口终端接收的信息，上面提示了有sd卡插入，同时设备节点是mmcblk0



mmc 应该是 sd 的前身，所以sd与mmc 的驱动通用，于是Linux 就把设备节点名称延续下来了。

blk表示这是块设备，随后跟的数字是该设备的编号blk0，表示编号为0的块设备，区分块设备用的。

p表示是分区，p1 表示这是第一个分区。

根据以上内容，我们应该就能看出mmcblk0，mmcblk0p1表示什么意思了。

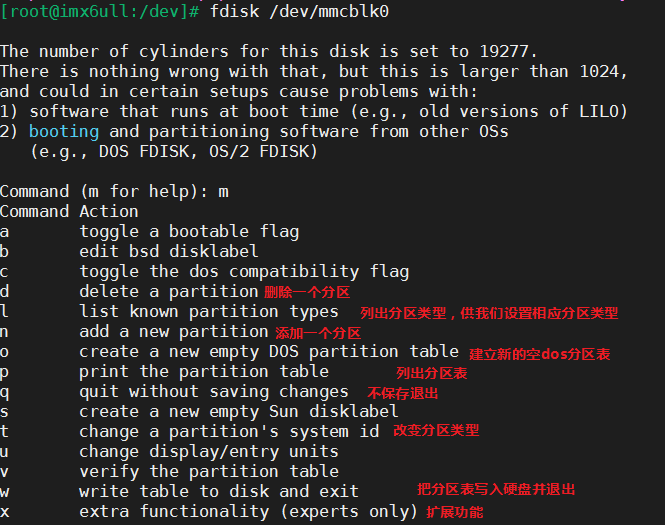
mmcblk0表示整个sd卡设备

mmcblk0p1表示这是sd卡设备的第一个分区。

#### 15.1.3 分区

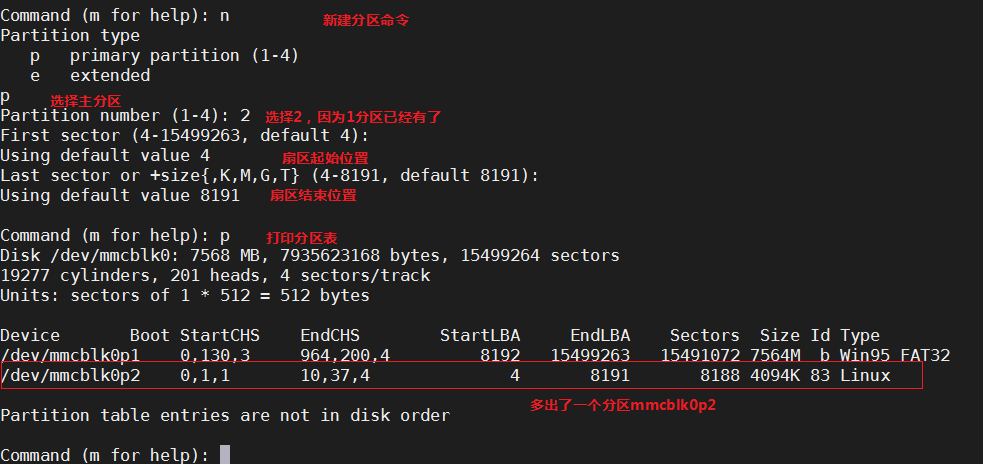
我们使用fdisk工具来分区，

执行fdisk /dev/mmcblk0后，输入m（进入菜单功能选项）进入 fdisk 画面：



常用的是：d l m p q t w命令

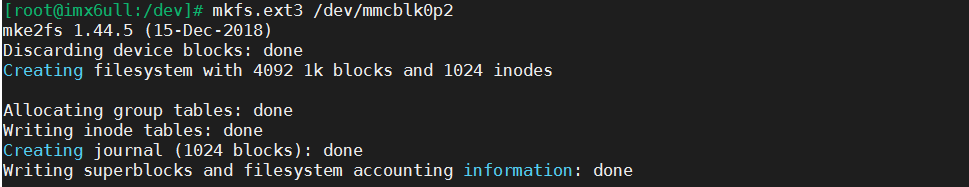
我们尝试利用剩余空间增加一个分区



此时按下w 就可以将分区信息存储到分区表中，并离开 fdisk；如果你决定这样的操作不对，也可以直接按下 q 取消刚刚的fdisk工具的所有动作，并退出fdisk。

#### 15.1.4 格式化并挂载

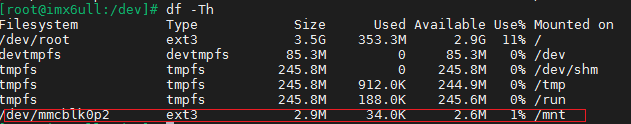
上一小节，我们新建立了一个分区，我们的第二个分区为 /dev/mmcblk0p2 ，分区类型为 Linux ,此时我们可以mkfs.ext3 /dev/mmcblk0p2对其进行格式化



此时我们可以用mount命令挂载该分区到我们想要的目录

mount -t ext3 /dev/mmcblk0p2 /mnt/

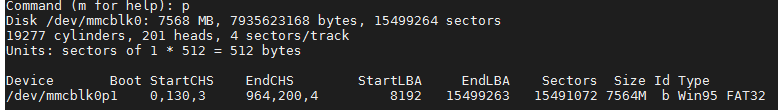
通过df –Th命令查看是否挂载成功。



当然我们也可以挂载mmcblk0p1这个分区，也是先指定某种文件系统格式化该分区，然后再用mount命令指定挂载的格式的同时挂载，最后df –Th命令查看是否挂载成功。

U盘的分区，格式化，挂载等操作和sd卡一致，只是将/dev/mmcblk0 变成了/dev/sda1。

#### 15.1.5 介绍分区表



上图是利用fdisk 里的p功能字，打印分区表功能显示的。

Device：表示这是哪一个分区，这里表示是mmcblk0的的一个分区；

Boot：表示启动引导标志；

StartCHS：分区开始的柱面、磁头、扇区；

EndCHS：分区结束的柱面、磁头、扇区；

StartLBA：逻辑块地址起始位置；

EndLBA： 逻辑块地址结束位置；

Sectors： 扇区数量；

Size：分区大小。

Id与Type：成对出现，id表示文件系统类型编号，type表示文件系统类型标记

注：这里的Id与Type其实只是做个标记，并没有实际变成我们想要的文件系统格式，因此如果需要挂载时，还需要用mkfs相关命令的格式化为对应文件系统格式后再挂载。

### 15.2 自动挂载U盘

#### 15.2.1 udev规则

udev是Linux（linux2.6内核之后）默认的设备管理工具。udev 以守护进程的形式运行，通过侦听内核发出来的 uevent 来管理 /dev目录下的设备文件。通过udev编写对应规则，实现设备节点变化时做出规定的动作，例如本节所说的热拔插实验、

udev常用的规则我们了解下。

1）规则文件中以 "#" 开头的行以及空行将被忽略；

2）规则文件必须以 .rules 作为后缀名，否则将被忽略；

3）规则文件分别位于： 系统规则目录(/usr/lib/udev/rules.d)、 运行时规则目录(/run/udev/rules.d)、 本机规则目录(/etc/udev/rules.d)，/etc/ 的优先级最高、 /run/ 的优先级居中、 /usr/lib/ 的优先级最低，也就是说我们可以在本机规则目录(/etc/udev/rules.d)下添加新的规则，它优先级最高，如果存有低优先级的同类规则，也会被高优先级的新规则替换掉，规则文件开头的数字越小它的执行顺序越靠前。

4）"键"有两种类型：匹配与赋值。 如果某条规则的所有匹配键的值都匹配成功，那么就表示此条规则匹配成功， 也就是此条规则中的所有赋值键都会被赋予指定的值。

匹配类：

①“==”等于；

②“!=”不等于；

③“ACTION”匹配事件的动作，例如"add"表示插入一个设备；

④“KERNEL”匹配设备的内核名称，如sda；

⑤“SUBSYSTEM” 所属的子系统。例如"sound"或"net"等。

赋值类：

①“=”，为键赋予指定的值；

②“LABEL”设置一个可用作 GOTO 跳转目标的标签；

③“GOTO” 跳转到下一个匹配的 LABEL 标签所在的规则；

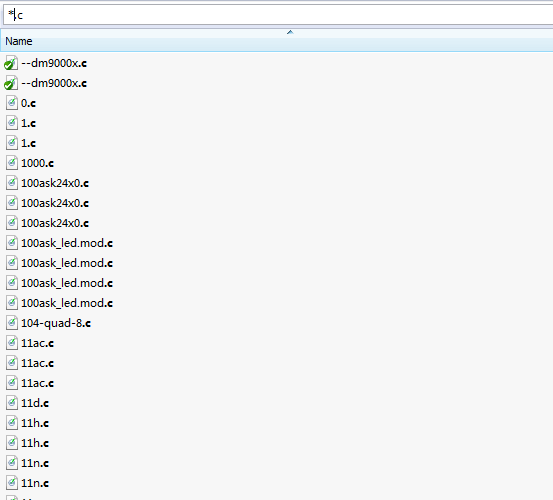
④“RUN”{类型}对于每一个设备事件来说，在处理完规则之后，都可以再接着执行一个程序列表(默认为空)。 不同的"类型"含义如下："program"一个外部程序， 如果是相对路径， 那么视为相对于 /usr/lib/udev 目录。 否则必须使用绝对路径。如果未明确指定"类型"， 那么这是默认值。"builtin"与 program 类似，但是仅用于表示内置的程序。程序名与其参数之间用空格分隔。 如果参数中含有空格，那么必须使用单引号(')界定。仅可使用运行时间非常短的前台程序， 切勿设置任何后台守护进程或者长时间运行的程序

如需更加详细关于udev内容可以参考这两篇文章《udev 中文手册 [金步国]》，《Writing udev rules.html》文章链接已保存在本章对应目录下。

#### 15.2.2 正则表达式

在udev的设备匹配上用到正则表达式，我们大致了解下，如需更加详细的内容可以参考这篇文章《正则表达式30分钟入门教程》，文章链接已保存在本章对应目录下。

①在电脑上查找文件，如果需要找‘.c’文件，我们通常会用‘\*.c’就可以查找全部的‘.c’文件，如下图



这里使用的 \* 就是通配符，表示任意字符。

②如果我们想要更加精确的表达的话就无能为力了，因此引入了正则表达式。

·：表示任意字符（换行符除外）；

\*：表示重复0次或更多次；

+：表示重复1次或更多次；

？：表示重复0次或1次；

[~]：表示这里面的字符里的某一个，例如[abc]表示abc中的其中一个，[1-9]表示1至9的其中一个。

例如leds?[1-3]，其中s？表示s出现0次或者1次，[1-3]？表示1至3的某一个，出现0次或1次，根据以上信息匹配的结果就是leds，led1，led2，led3。我们通过一个语句就匹配了4个设备，多么简练。

#### 15.2.3 自动挂载U盘

①创建用于挂载U盘的目录

mkdir /mnt/usb –p

②在/etc/udev/rules.d目录下添加用于检测U盘插入规则（add），终端下执行以下命令创建第一个U盘插入规则。

vim /etc/udev/rules.d/11-add-usb.rules

在11-add-usb.rules中添加如下内容：

ACTION!="add",GOTO="END"

KERNEL=="sda[0-9]",RUN+="/etc/mount-usb.sh %k"

LABEL="END"

上面的内容意思是：如果不是add添加事件，那么就跳到END标签处结束，如果是add事件那么执行以下规则，如果添加的设备是sda0至sda9中任意一个，那么就执行RUN后面的内容，此处是一个脚本，一个挂载U盘的脚本，它还带有个参数%k，表示kernel的值。

然后在/etc/目录下创建mount-usb.sh脚本

在mount-usb.sh中添加如下内容：

#!/bin/sh  
 mount  -t vfat /dev/$1 /mnt/usb  
 sync

上面的内容意思是：#!/bin/sh表示是脚本文件，按脚本文件解析，mount -t vfat /dev/$1 /mnt/usb表示按vfat格式将/dev/$1挂载到/mnt/usb目录下，其中这里的$1就是之前规则里传入进来的%k，也就是kernel值。Sync表示同步U盘数据。

最后记得给脚本文件添加执行权限。

chmod +x /etc/mount-usb.sh

③在/etc/udev/rules.d目录下添加用于检测U盘移出规则（remove），终端下执行以下命令创建第U盘移出规则。

vim /etc/udev/rules.d/11-remove-usb.rules

在11-remove-usb.rules中添加如下内容：

ACTION!="remove",GOTO="END"

KERNEL=="sda[0-9]",RUN+="/etc/umount-usb.sh %k"

LABEL="END"

上面的内容意思是：如果不是remove添加事件，那么就跳到END标签处结束，如果是remove事件那么执行以下规则，如果移出的设备是sda0至sda9中任意一个，那么就执行RUN后面的内容，此处是一个脚本，一个卸载U盘的脚本，它还带有个参数%k，表示kernel的值。

然后在/etc/目录下创建umount-usb.sh脚本

在umount-usb.sh中添加如下内容：

#!/bin/sh

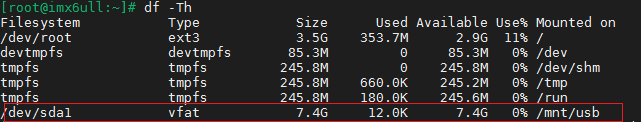
sync  
 umount /mnt/usb

上面的内容意思是：#!/bin/sh表示是脚本文件，按脚本文件解析，先同步U盘数据，然后再卸载，其中这里的$1就是之前规则里传入进来的%k，也就是kernel值。

最后记得给脚本文件添加执行权限。

chmod +x /etc/umount-usb.sh

④设置好规则后，当我们插入U盘，执行df –Th就会显示如下图



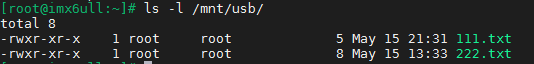
表明/dev/sda1已经成功挂载在/mnt/usb目录下了。

注1： vfat格式也就是fat32格式，使用vfat格式是由于windows操作系统与linux操作系统都支持，这样U盘就可以在windows操作系统与linux操作系统之间作为交换文件的介质。

注2：Linux 系统中欲写入U盘等存储介质内时，有的时候为了效率起见，会写到 filesystem buffer 中，这个 buffer 是一块记忆体空间，如果欲真正的写入U盘等存储介质内需要执行sync 指令，它会将存于 buffer 中的数据强制写入U盘等存储介质内，这也是为什么脚本里在要加入sync的原因。

### 15.3 挂载后就是一般的读写文件

成功挂载后，我们就可以在挂载目录下进行创建，修改等文件操作，就等同于操作U盘上的文件。

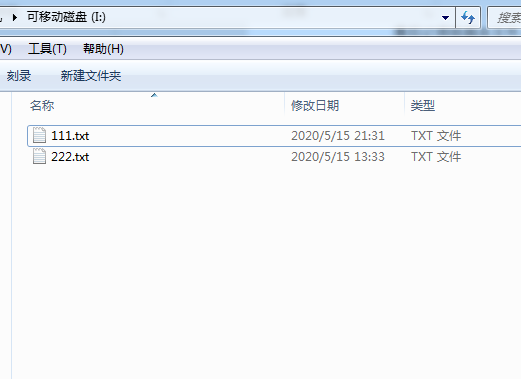


如上图，我们在终端使用以下命令创建两个文本文件。

touch /mnt/usb/111.txt

touch /mnt/usb/222.txt

然后再拔出U盘，插到电脑中查看是否存在111.tx与222.txt



我们可用在上面添加些内容，然后再插入开发板，查看电脑上修改的内容在开发板上是否显示。在111.txt中添加abc，在222.txt中添加100ask，保存，弹出U盘，插入开发板。 通过cat命令查看内容，如下图



显示的内容正确，以后我们就可以通过U盘把电脑的东西往开发板上搬了。