

《嵌入式系统》

（第八讲）

厦门大学信息学院软件工程系 曾文华

2024年10月29日

第8章 文件系统

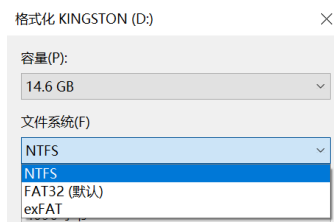
- 8.1 嵌入式文件系统简介
- 8.2 嵌入式Linux文件系统框架
- 8.3 JFFS2嵌入式文件系统
- 8.4 根文件系统

8.1 嵌入式文件系统简介

• 8.1.1 Linux文件系统简介

- 每个操作系统使用的文件系统并不相同，例如，Windows 98 以前的微软操作系统使用 FAT（FAT16）文件系统，Windows 2000 以后的版本使用 NTFS 文件系统，而 Linux 的正统文件系统是 **Ext2**
 - **FAT**: File Allocation Table，文件配置表
 - **FAT16**使用了16位的空间来表示每个扇区(Sector)配置文件的情形，故称之为FAT16
 - **NTFS**: New Technology File System，新技术文件系统
 - **Ext2**: second Extended file system，第二代扩展文件系统

格式化磁盘



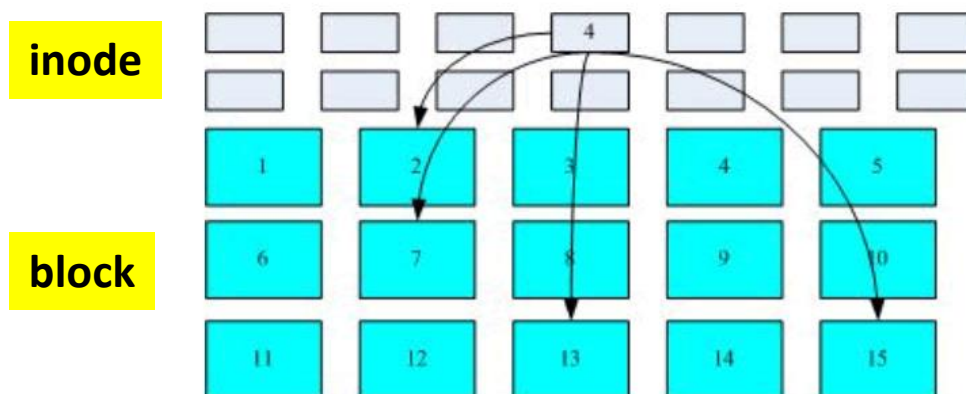
FAT32是从FAT和FAT16发展而来的，优点是稳定性和兼容性好，能充分兼容Win 9X及以前版本，且维护方便。缺点是安全性差，单个文件也只能支持最大4GB。

exFAT的意思是扩展FAT，即扩展文件分配表。是微软在2006年引入的文件系统，解决了其前身FAT32文件系统不支持4G及其更大文件的问题，在exFAT文件系统中单个文件的最大容量可以达到16 EB。

Linux支持的常见文件系统

| 文件系统 | 描述 |
|---------|---|
| Ext | Linux 中最早的文件系统，由于在性能和兼容性上具有很多缺陷，现在已经很少使用 |
| Ext2 | 是 Ext 文件系统的升级版本，Red Hat Linux 7.2 版本以前的系统默认都是 Ext2 文件系统。于 1993 年发布，支持最大 16TB 的分区和最大 2TB 的文件 (1TB=1024GB=1024x1024KB) |
| Ext3 | 是 Ext2 文件系统的升级版本，最大的区别就是带日志功能，以便在系统突然停止时提高文件系统的可靠性。支持最大 16TB 的分区和最大 2TB 的文件 |
| Ext4 | 是 Ext3 文件系统的升级版。Ext4 在性能、伸缩性和可靠性方面进行了大量改进。Ext4 的变化可以说是翻天覆地的，比如向下兼容 Ext3、最大 1EB 文件系统和 16TB 文件、无限数量子目录、Extents 连续数据块 概念、多块分配、延迟分配、持久预分配、快速 FSCK、日志校验、无日志模式、在线碎片整理、inode 增强、默认启用 barrier 等。它是 CentOS 6.3 的默认文件系统 |
| xfs | 被业界称为最先进、最具有可升级性的文件系统技术，由 SGI 公司设计，目前最新的 CentOS 7 版本默认使用的就是此文件系统。 |
| swap | swap 是 Linux 中用于交换分区的文件系统（类似于 Windows 中的虚拟内存），当内存不够用时，使用交换分区暂时替代内存。一般大小为内存的 2 倍，但是不要超过 2GB。它是 Linux 的必需分区 |
| NFS | NFS 是网络文件系统（Network File System）的缩写，是用来实现不同主机之间文件共享的一种网络服务，本地主机可以通过挂载的方式使用远程共享的资源 |
| iso9660 | 光盘的标准文件系统。Linux 要想使用光盘，必须支持 iso9660 文件系统 |
| fat | 就是 Windows 下的 fat16 文件系统，在 Linux 中识别为 fat |
| vfat | 就是 Windows 下的 fat32 文件系统，在 Linux 中识别为 vfat。支持最大 32GB 的分区和最大 4GB 的文件 |
| NTFS | 就是 Windows 下的 NTFS 文件系统，不过 Linux 默认是不能识别 NTFS 文件系统的，如果需要识别，则需要重新编译内核才能支持。它比 fat32 文件系统更加安全，速度更快，支持最大 2TB 的分区和最大 64GB 的文件 |
| ufs | Sun 公司的操作系统 Solaris 和 SunOS 所采用的文件系统 |
| proc | Linux 中基于内存的虚拟文件系统，用来管理内存存储目录 /proc |
| sysfs | 和 proc 一样，也是基于内存的虚拟文件系统，用来管理内存存储目录 /sysfs |
| tmpfs | 也是一种基于内存的虚拟文件系统，不过也可以使用 swap 交换分区 |

- 通常情况下，文件系统会将文件的实际内容和属性分开存放：
 - 文件的属性保存在 **inode** 中（i 节点）中，每个 **inode** 都有自己的编号。每个文件各占用一个 **inode**。不仅如此，**inode** 中还记录着文件数据所在 **block** 块的编号。
 - 文件的实际内容保存在 **block** 中（数据块），每个 **block** 都有属于自己的编号。当文件太大时，可能会占用多个 **block** 块。
 - 另外，还有一个 **super block**（超级块）用于记录整个文件系统的整体信息，包括 **inode** 和 **block** 的总量、已经使用量和剩余量，以及文件系统的格式和相关信息等。



假设某文件的权限和属性信息存放到 **inode 4** 号位置，这个 **inode** 记录了实际存储文件数据的 **block** 号有 4 个，分别为 2、7、13、15

• 8.1.2 嵌入式文件系统简介

– 1、嵌入式操作系统的文件系统的设计目标

- ① 使用简单方便
- ② 安全可靠
- ③ 实时响应
- ④ 接口标注的开放性和可移植性
- ⑤ 可伸展性和可配置性
- ⑥ 开放的体系结构
- ⑦ 资源有效性
- ⑧ 功能完整性
- ⑨ 热插拔
- ⑩ 支持多种文件类型

– 2、一些流行的嵌入式文件系统

QNX是一种商用的遵从POSIX规范类Unix实时操作系统，目标市场主要是面向嵌入式系统

- **QNX**嵌入式操作系统：

- ① **POSIX文件系统**：Portable Operating System Interface of UNIX，可移植操作系统接口
- ② **SMB文件系统**：Server Message Block，服务器报文块协议
- ③ **FAT文件系统**：File Allocation Table，文件分配表
- ④ **CD-ROM文件系统**

VxWorks 操作系统是美国WindRiver公司于1983年设计开发的一种嵌入式实时操作系统（RTOS）

- **VxWorks**嵌入式操作系统：

- ① **FFS**：Fast File System，快速文件系统
- ② **SCSI**：Small Computer System Interface，小型计算机系统接口
- ③ **FAT**：File Allocation Table，文件分配表
- ④ **RT11FS**：Real Time 11 File System，实时文件系统
- ⑤ **TAPEFS**：TAPE File System，磁带文件系统

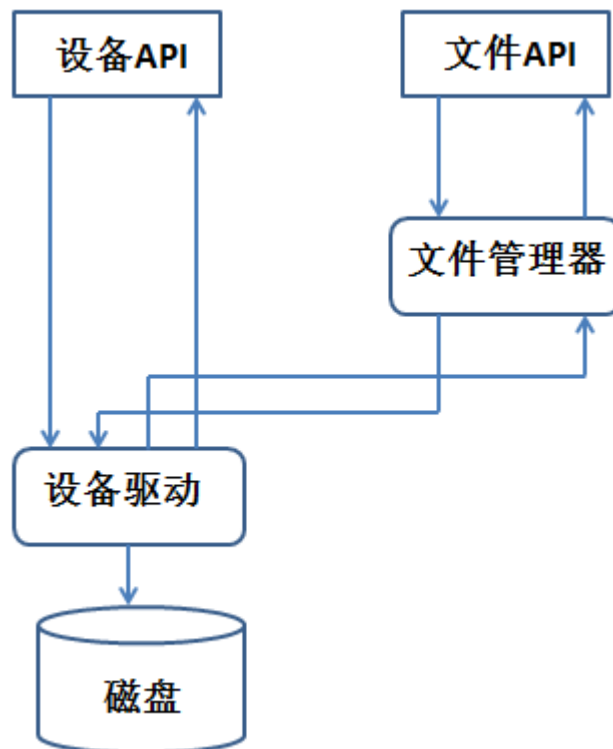
8.2 嵌入式Linux文件系统框架

- 传统文件系统

- 采用设备API，经过设备驱动，直接去访问磁盘。
- 或者，采用文件API，经过文件管理器，以及设备驱动，去访问磁盘。

设备API函数：

- ① open()
- ② close()
- ③ read()
- ④ write()
- ⑤ lseek()：移动文件读/写指针
- ⑥ ioctl()

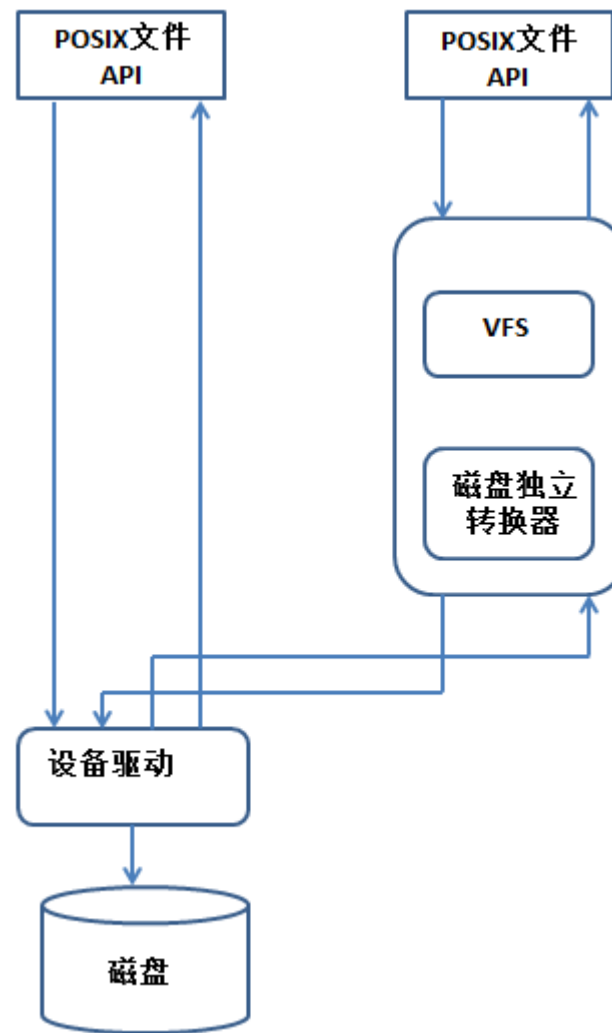


• Linux文件系统

- 采用POSIX文件API，经过设备驱动，直接去访问磁盘。
- 或者，采用POSIX文件API，经过**VFS**和**磁盘独立转换器**，以及设备驱动，去访问磁盘。

POSIX: Portable Operating System Interface of UNIX，可移植操作系统接口

VFS: Virtual File Systems，虚拟文件系统



8.3 JFFS2嵌入式文件系统

- **JFFS**: Journalling Flash File System, **闪存设备日志型文件系统**, 最初是由瑞典的 Axis Communication AB 公司开发, 其目的是作为嵌入式系统免受宕机和断电危害的文件系统。
- **JFFS2**: Journalling Flash File System Version2, **闪存日志型文件系统第2版**, 其功能就是管理在**MTD设备**上实现的日志型文件系统。除了提供具有断电可靠性的日志结构文件系统, JFFS2还会在它管理的MTD设备上实现“损耗平衡”和“数据压缩”等特性。

MTD: Memory Technology Device, **内存技术设备**。是用于访问memory设备（ROM、flash）的Linux的子系统。MTD的主要目的是为了使得新的memory设备的驱动更加简单, 为此它在硬件和上层之间提供了一个抽象的接口。MTD的所有源代码在/drivers/mtd子目录下。CFI接口的MTD设备分为四层（从设备节点直到底层硬件驱动），这四层从上到下依次是：设备节点、MTD设备层、MTD原始设备层和硬件驱动层。

- JFFS2的数据结构:

```
struct jffs2_unknown_node
```

```
{
```

```
    _u16 magic;
```

```
//节点类型的补充
```

```
    _u16 nodetype;
```

```
//节点类型
```

```
    _u32 totlen;
```

```
//节点总长度
```

```
    _u32 hdr_crc;
```

```
//CRC校验码
```

```
}
```

- JFFS2数据结构的内存表示:

MSB: Most
Significant Bit,
最高有效位

MSB

LSB

| | |
|--------------|---------------|
| _u16 magic | _u16 nodetype |
| _u32 totlen | |
| _u32 hdr_crc | |

LSB: Least
Significant Bit,
最低有效位

• 8.3.1 目录节点的定义

```
struct jffs2_raw_direct
```

```
{
```

```
    _u16 magic;
```

```
//节点类型的补充
```

```
    _u16 nodetype;
```

```
//节点类型
```

```
    _u32 totlen;
```

```
//节点总长度
```

```
    _u32 hdr_crc;
```

```
//CRC校验码
```

```
    _u32 pino;
```

```
//上层目录节点的标号
```

```
    _u32 version;
```

```
    _u32 ino;
```

```
//节点编号
```

```
    _u32 mctime;
```

```
//创建时间
```

```
    _u8 nsize;
```

```
//大小
```

```
    _u8 unused[2];
```

```
    _u32 nod_crc;
```

```
//校验码
```

```
    _u32 name_crc;
```

```
    _u8 name[0];
```

```
//名称
```

```
}
```

• 8.3.2 数据节点

struct jffs2_raw_inode

```
{
    _u16 magic;                //节点类型的补充
    _u16 nodetype;             //节点类型
    _u32 totlen;                //节点总长度
    _u32 hdr_crc;              //CRC校验码
    _u32 pino;                  //上层目录节点的标号
    _u32 version;
    _u32 ino;
    _u32 mode;                  //文件的类型
    _u32 uid;
    _u32 gid;
    _u32 isize;                //实际长度
    _u32 atime;
    _u32 mtime;
    _u32 ctime;
    _u32 offset;               //对应数据文件的起始位置
    _u32 csize;                //压缩数据的长度
    _u32 dsize;                //数据有效长度
    _u8 compr;                  //当前压缩算法
    _u8 usercompr;              //当前指定的压缩算法
    _u16 flage;                 //标志位
    _u32 data_crc;              //数据校验码
    _u32 name[0];              //名称
}
```

• 8.3.3 内存使用

- JFFS2中的i节点信息并没有全部存放在内存里面，存放在内存中的节点信息是一个缩小尺寸的数据节点结构体（**struct jffs2_raw_node_ref**）：

```
struct jffs2_raw_node_ref
{
    struct jffs2_raw_node_ref * next_in_ino;           //链表指针
    struct jffs2_raw_node_ref next_phys;              //在物理上相邻的块
    _u32 flash_offset;                                //在Flash块中的偏移
    _u32 totlen;
}
```

- 通过使用**struct jffs2_inode_cache**结构体，管理**struct jffs2_raw_node_ref**信息。
- 通过使用**struct jffs2_sb_info**结构体，管理所有的节点链表和Flash块。

• 8.3.4 JFFS3简介

- **JFFS3**: Journalling Flash File System Version3, 闪存日志型文件系统第3版
- JFFS3支持大容量Flash: 大于1TB
- JFFS3是将索引信息存放在Flash上, 而JFFS2则将索引信息保存在内存中
- JFFS3的基本结构借鉴了ReiserFS4的设计思想, 整个文件系统就是一个B+树

ReiserFS4: 是一种新型的文件系统, 它通过一种与众不同的方式——完全平衡树结构来容纳数据, 包括文件数据, 文件名以及日志支持。**ReiserFS4**还支持海量磁盘和磁盘阵列, 并能在上面继续保持很快的搜索速度和很高的效率。

B+树: 是一种树数据结构, 通常用于数据库和操作系统的文件系统中。**B+树**的特点是能够保持数据稳定有序, 其插入与修改拥有较稳定的对数时间复杂度。**B+树**元素自底向上插入, 这与二叉树恰好相反。

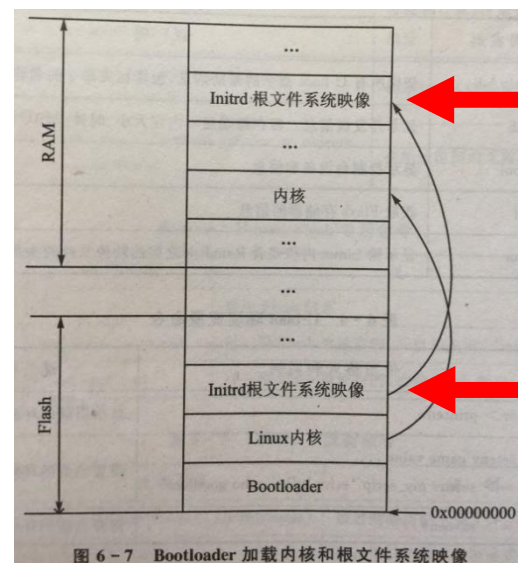
8.4 根文件系统

• 8.4.1 什么是根文件系统

– 系统挂载的第一个文件系统就是**根文件系统**

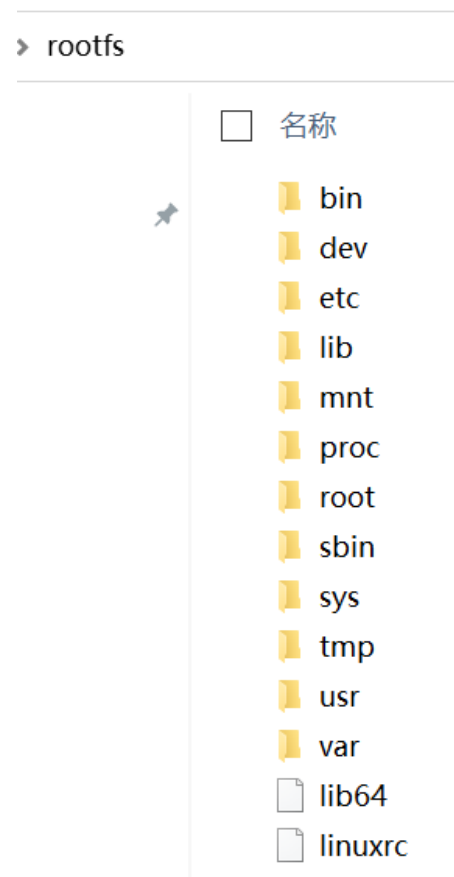
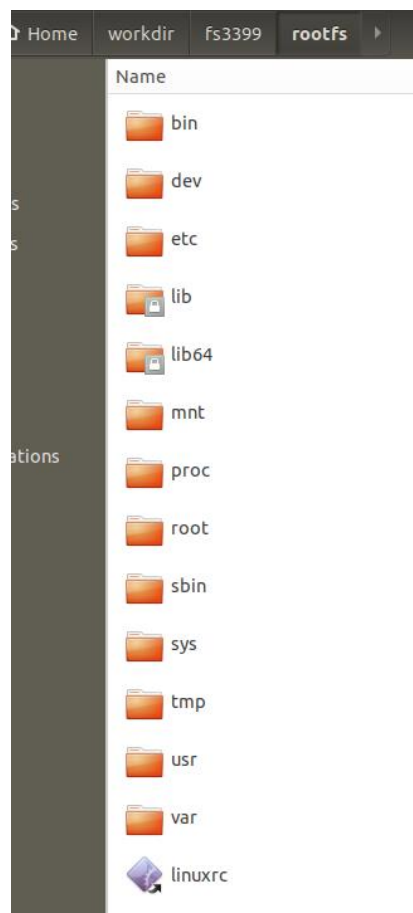
– 根文件系统**顶层目录**:

- ① **bin**: 用户命令所在目录
- ② **sbin**: 系统管理员命令目录
- ③ **usr**: 共享的文件
- ④ **proc**: 虚拟文件系统, 用来显示内核及进程信息
- ⑤ **dev**: 硬件设备文件及其它特殊文件
- ⑥ **etc**: 系统配置文件, 包括启动文件等
- ⑦ **lib**: 链接库文件目录
- ⑧ **boot**: 引导加载程序使用的静态文件
- ⑨ **home**: 多用户主目录
- ⑩ **mnt**: 装配点, 用于装配临时文件系统或其他的文件系统
- ⑪ **opt**: 附加的软件套件目录
- ⑫ **root**: 用户主目录
- ⑬ **tmp**: 临时文件目录
- ⑭ **var**: 监控程序和工具程序所存放的可变数据

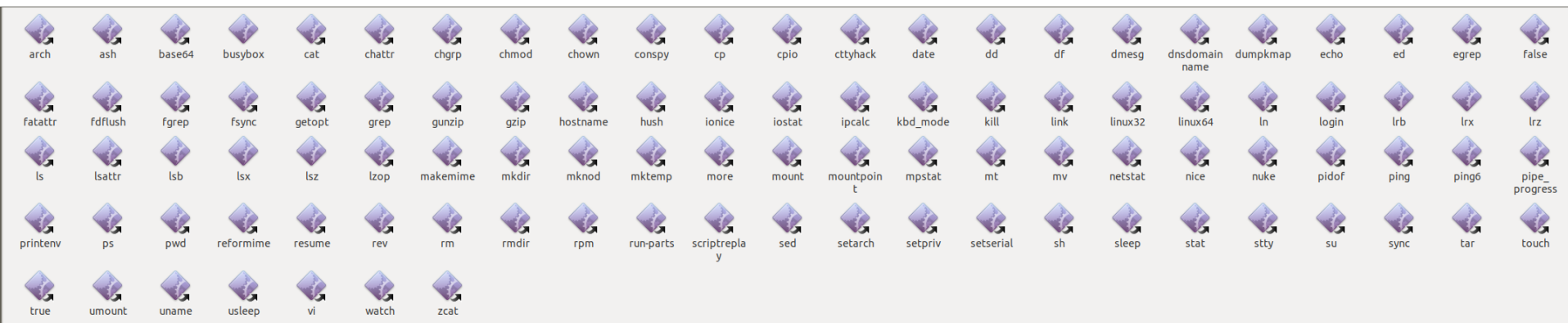


- **Ubuntu的根文件系统（在Ubuntu的“终端”上执行）：**
 - **linux@linux-pc:/\$ cd /**
 - **linux@linux-pc:/\$ ls**
 - bin boot cdrom dev etc home initrd.img initrd.img.old lib lib32 lib64 libx32 media
 - mnt opt proc root run sbin snap source srv sys tftpboot tmp usr var vmlinuz vmlinuz.old
 - **linux@linux-pc:/\$**
- **实验箱的根文件系统（在Xshell上执行）：**
 - **linux@localhost:~\$ cd /**
 - **linux@localhost:/\$ ls**
 - bin dev home lost+found mnt proc run snap sys tmp var
 - boot etc lib media opt root sbin srv system usr
 - **linux@localhost:/\$**

- 实验箱的根文件系统位于Ubuntu的 **/home/linux/fs3399/rootfs/** 目录



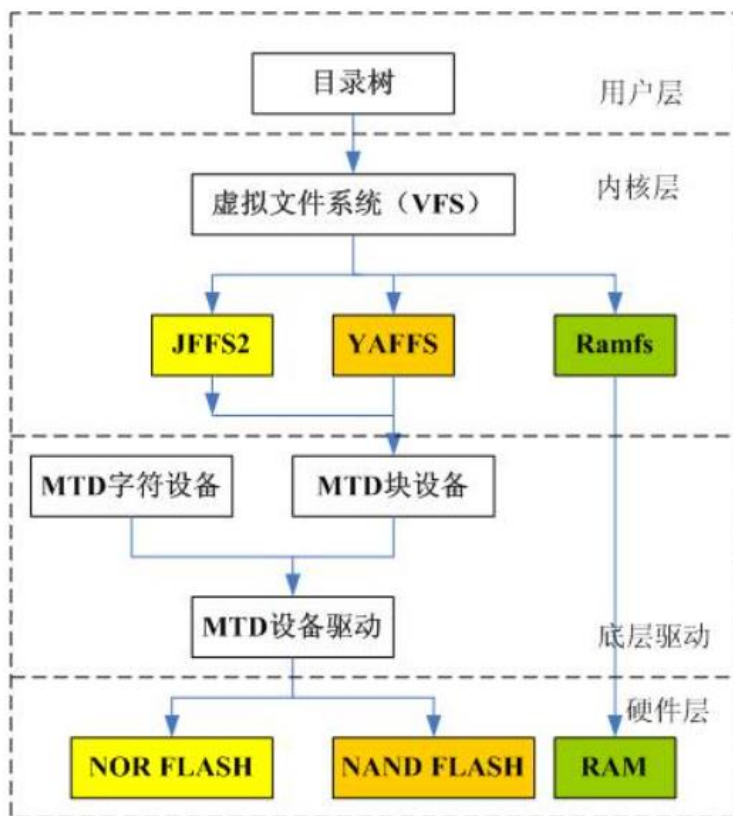
bin 目录



• 8.4.2 建立JFFS2根文件系统

– Linux下常用文件系统结构：

- 采用JFFS2作为根文件系统



Linux下常用文件系统结构

- **VFS: Virtual File Systems**, 虚拟文件系统。
- **JFFS2: Journalling Flash File System Version2**, 闪存日志型文件系统第2版, 其功能就是管理在MTD设备上实现的日志型文件系统。除了提供具有断电可靠性的日志结构文件系统, JFFS2还会在它管理的MTD设备上实现“损耗平衡”和“数据压缩”等特性。
- **YAFFS: Yet Another Flash File System**, 是专为嵌入式系统使用NAND型闪存而设计的一种日志型文件系统。
- **Ramfs**: 基于RAM的文件系统。
- **MTD: Memory Technology Device**, 内存技术设备。是用于访问memory设备 (ROM、flash) 的Linux的子系统。

– 手工建立JFFS2文件系统的步骤：

- ① 准备制作JFFS2根文件系统的工具： **mkfs.jffs2**
- ② 建立根文件系统的目录
- ③ 编译**busyBox**
- ④ 复制动态链接库到**lib**目录中
- ⑤ 创建**/etc/init.d/rcS**、**/etc/profile**、**/etc/fstab**、**/etc/inittab**文件，并且复制主机中的**/etc/passwd**、**/etc/shadow**、**/etc/group**文件到相应的目录中
- ⑥ 移植**bash**，将其复制到**/bin**目录中
- ⑦ 执行“**mkfs.jffs2 -r ./rootfs -o rootfs.jffs2 -n -e 0x20000**”，生成JFFS2根文件系统镜像

第16章 基于 busybox 的最小文件系统制作

16.1 busybox 源码编译及安装

实验箱的根文件系统（最新文件系统）制作请参考“**实验指导书（嵌入式Linux应用与驱动开发）.pdf**”的第16章

可以从 <http://busybox.net/downloads/> 网站下载 busybox-1.29.3 源码用于制作 Linux 文件系统，为了方便，已将源码放进了光盘。

建立源码目录

将【华清远见-嵌入式 ARM 实验箱资料-I-FS3399\程序源码\Linux 最小文件系统\busybox 源码】下的 busybox-1.29.3.tar.bz2 拷贝至虚拟机目录，并进入该目录下。

| | |
|---|-----------------------|
| <code>mkdir ~/hcyj/busybox -p</code> | //在家目录下创建文件夹 |
| <code>cd ~/hcyj/busybox</code> | //进入刚创建的目录下 |
| <code>cp /mnt/hgfs/share/busybox-1.29.3.tar.bz2 ./</code> | //将 busybox 源码拷贝到此目录下 |
| <code>tar xvf busybox-1.29.3.tar.bz2</code> | //解压文件 |
| <code>cd busybox-1.29.3</code> | //进入解压后文件夹 |

配置 busybox 源码：

```
vi Makefile
```

将顶层目录下的 Makefile 文件中的 CROSS_COMPILE 字段修改为“/home/linux/toolchain/6.4-aarch64/bin/aarch64-linux-gnu-”

16.2 添加主要系统启动文件

创建其他需要的目录：

```
mkdir dev etc mnt proc var tmp sys root
```

添加库：

将工具链中的库拷贝到_install 目录下：

```
cp -a /home/linux/toolchain/6.4-aarch64/aarch64-cortexa53-linux-gnu/sysroot/lib* .
```

删除静态库：

```
rm lib/*.a
```

添加系统启动文件：

在 etc 下添加文件 inittab，文件内容如下：

注意：修改文件均为_install 目录下

etc/inittab

示例代码 16-1 添加内容

```
1 #this is run first except when booting in single-user mode.
2 ::sysinit:/etc/init.d/rcS
3 # /bin/sh invocations on selected ttys
4 # start an "askfirst" shell on the console (whatever that may be)
5 ::askfirst:-/bin/sh
```

16.3 制作最小文件系统烧写镜像

前面我们做的最小文件系统是通过 NFS 挂载的方式，放到我们的开发板上，本节实验我们制作最小文件系统，烧写进我们的开发板。

1. 制作 rootfs.img

dd: 用指定大小的块拷贝一个文件，并在拷贝的同时指定转换成想要的格式。

if=文件名: 输入文件名，缺省为标准输入。

of=文件名: 输出文件名，缺省为标准输出。

192

bs=bbytes: 同时设置输入/输出的块大小为 bytes 个字节。

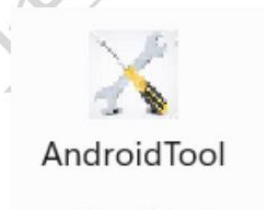
count=blocks: 仅拷贝 blocks 个块，块大小等于指定的 bs 字节数。

```
cd ~/hcyj
```

```
dd if=/dev/zero of=rootfs.img bs=1M count=300
```

```
linux@linux-pc:~/hcyj$ dd if=/dev/zero of=rootfs.img bs=1M count=300
300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 1.09356 s, 288 MB/s
linux@linux-pc:~/hcyj$ ls
```


16.4 烧写最小文件系统



1. 打开我们的烧写工具 **AndroidTool**。
2. 连接开发板



图 16-7 接线

3. 按住开发板的 **VOL+**键，然后开发板上电。

确定下方显示“发现一个 LOADER 设备”。

勾选 rootfs 前的方框，地址为 0x0005A000，路径为 rootfs.img 所在的路径 + /rootfs.img。

rootfs.img 可使用上节制作的文件系统镜像，也可以直接在资料中获取，路径在【华清远见-嵌入式

小结

- 嵌入式Linux文件系统的框架
- 常见的几个嵌入式文件系统：
 - ① **JFFS2**: Journalling Flash File System Version2, 闪存日志型文件系统第2版
 - ② **JFFS3**: Journalling Flash File System Version3, 闪存日志型文件系统第3版
 - ③ **YAFFS**: Yet Another Flash File System, 是专为嵌入式系统使用NAND型闪存而设计的一种日志型文件系统
 - ④ **Cramfs**: Compressed ROM File System, 只读压缩的文件系统
 - ⑤ **Ramfs**: 基于RAM的文件系统
- 如何制作根文件系统（最小文件系统）？同学们可以按照实验指导书在自己的电脑上试一下！

进一步探索

- **Cramfs**根文件系统的制作：
 - 使用 **busybox** 生成文件系统中的命令部分
 - 使用**mkcramfs** 工具制作**Cramfs** 格式的根文件系统

Thanks