嵌入式操作系统

7 嵌入式Linux开发技术

陈香兰（xlanchen@ustc.edu.cn）

计算机应用教研室@计算机学院
嵌入式系统实验室@苏州研究院
中国科学技术大学
Fall 2014

December 4, 2014

Outline

Outline

Outline

建立目标板Linux的基本步骤

建立目标板Linux系统有4个重要的步骤：

1. 决定系统组件
   - Linux具有大量可选软件，应当为目标系统列出必须的功能清单
2. 配置并建立内核
   - 选择合适的Linux内核版本与适当的配置
   - 建立内核
3. 建立根文件系统
4. 设置引导软件与配置

Outline

开发嵌入式Linux系统最常用的主机类型

1. Linux工作站
   - 通常就是一台安装了某个标准的Linux发行套件的PC机，如Debian、Mandrake、Red Hat等。
   - 需约2、3G或更多的磁盘空间进行嵌入式Linux开发
   - 建议具有128或以上的RAM及交换空间
2. Unix工作站
   - 由于Linux与Unix非常相似，对Linux适用的通常对Unix也适用

开发嵌入式Linux系统最常用的主机类型

3. Windows工作站
   - 许多开发者比较习惯Windows平台，并希望在Windows平台上开发嵌入式Linux系统
   - Cygwin：
     可在Windows平台上建立跨平台开发工具链
   - VMWare/VirtualBox：
     可在虚拟环境中执行Linux，并在Linux上进行嵌入式Linux的开发

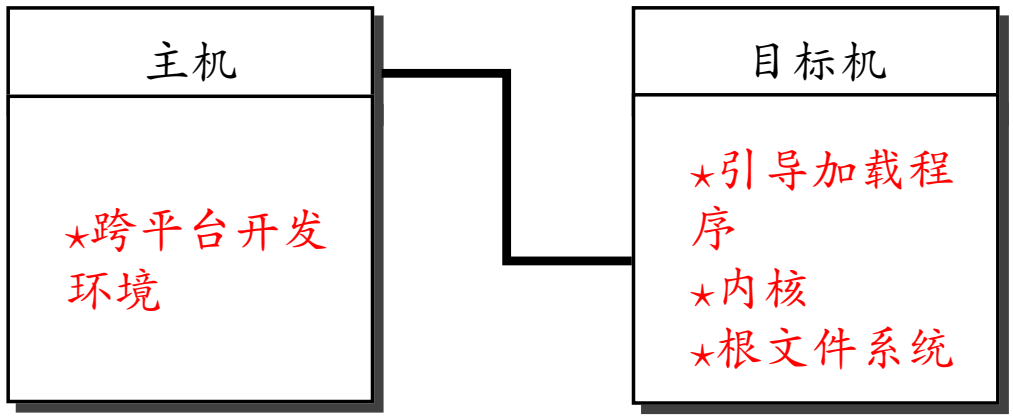
Outline

主机/目标机的开发体系结构

- 在嵌入式Linux系统开发中，存在3种主机/目标机开发体系结构
  1. 连接式
  2. 使用可移动存储设备
  3. 独立开发式

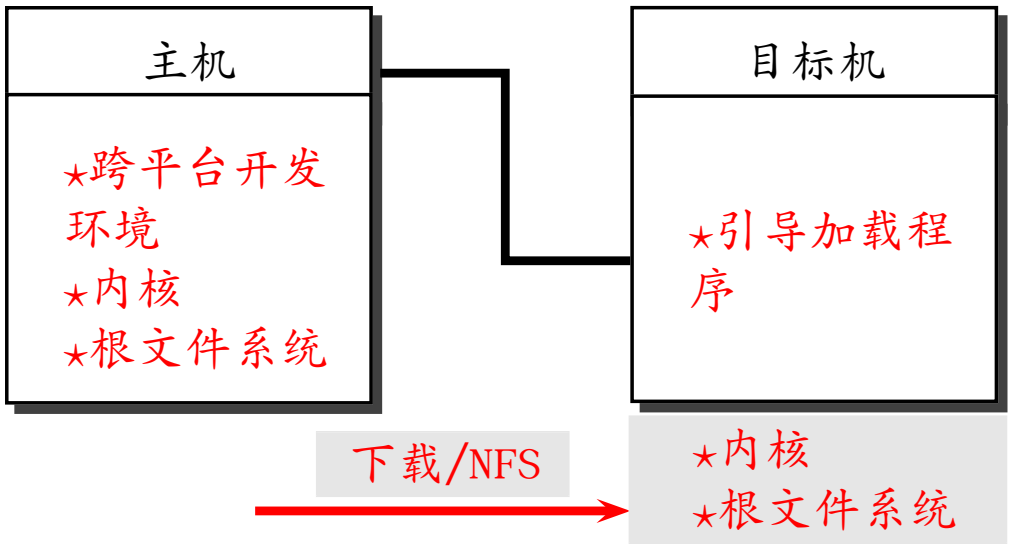
1、连接式 （the linked setup）

- 目标板和主机通过一个物理线路永久的连接在一起：
  如串行线或以太网连接
- 好处：
  目标代码的传送无需物理存储设备参与，只需要上述连接就足够了
- 在这种方式中，主机包含了跨平台开发环境，而目标板则包含了适当的引导加载程序、可用的内核以及最起码的根文件系统



1、连接式 （the linked setup）

- 另一种做法是，
  以远程组件来简化目标板的开发工作，例如
  - 通过TFTP下载内核
  - 此外，根文件系统还可以通过NFS安装，而不必在目标板中使用存储介质



TFTP：
Trivial File Transfer Protocol
小型文件传输协议
比FTP简单

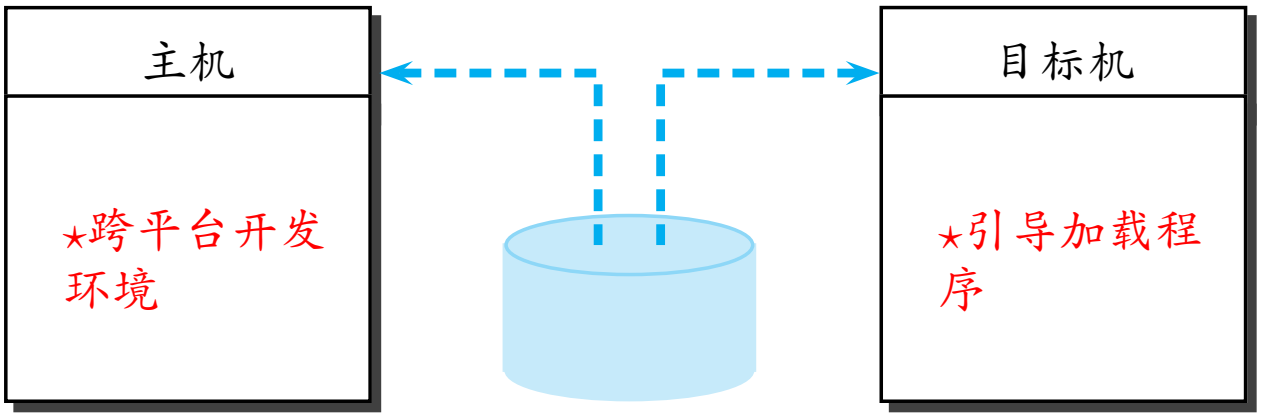
NFS：
Network File System
网络文件系统
允许访问远程计算机上的文件

1、连接式 （the linked setup）

- 还可以使用连接进行调试
  通常使用以太网连接进行下载功能，而使用RS232串口连接进行调试

2、使用可移动存储设备，
the removable storage setup

- 主机和目标板之间没有实际的连接。
  先由主机将数据写入存储设备，然后将存储设备转接到目标板，并使用该存储设备引导目标板
- 同样的，在主机上包含了跨平台开发环境。而目标板则只包含了最起码的引导加载程序。其余的组件被存放在可移动存储设备上。

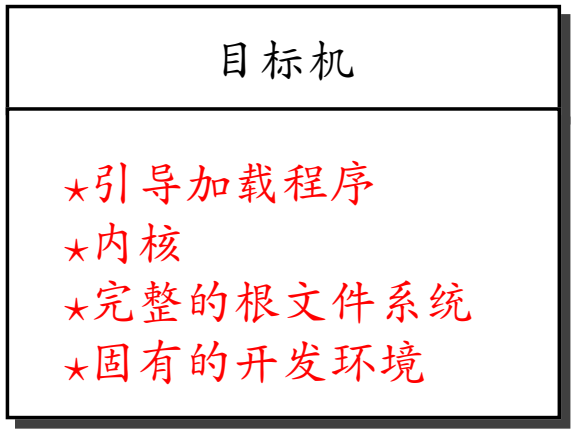


2、使用可移动存储设备，
the removable storage setup

- 一种通常的操作方式使用易插拔的flash芯片：
  - 先在主机上使用flash编程器将数据写入芯片，
  - 然后再将该芯片插入目标板上的插座中

3、独立开发系统

- 在这种设置中，目标板是个独立的开发系统，它包含了引导、操作以及开发额外软件所必须的任何软件。
- 不需要跨平台开发环境，不必在主机和目标板之间传送任何数据



- 适合以PC为主的高级嵌入式系统的开发

Outline

主机/目标板的调试方式

- 用来连接目标板与主机进行调试的接口基本上有3种类型：
  1. 串行线、
  2. 网络接口、
  3. 特殊的调试硬件，如BDM、JTAG

1、使用串行线进行调试

- 这是从主机对目标板进行调试的最简单的做法
- 缺点：
  - 串行连接的速度比较有限
  - 当嵌入式系统中只有一个串行串口，
    或者串行连接是嵌入式系统对外唯一的接口，
    那么就不可能在对系统调试的同时，以终端仿真器
    跟系统交互。

2、使用网络接口进行调试

- 这种方式与串行线连接相比，可以提供较高的带宽
- 由于可以在相同的物理网络连接上使用多重网络连接，可以兼顾调试与终端仿真交互
- 缺点：
  无法使用网络连接对Linux内核进行调试。
  因为网络协议栈本身在Linux内核里。
  相对而言，内核的调试通常可以通过串行连接来进行

3、使用特殊的调试硬件

- 通常会使用BDM或JTAG接口。
- 这些接口依靠的是CPU芯片中内嵌的BDM或JTAG特殊功能。
  - 只要将一个特殊的调试器连接到CPU上的JTAG或BDM相关管脚，就可以完全控制CPU的行为。
  - 因此，当遇到新的嵌入式目标板、或者对目标板上的Linux内核进行调试时，通常会使用JTAG和BDM

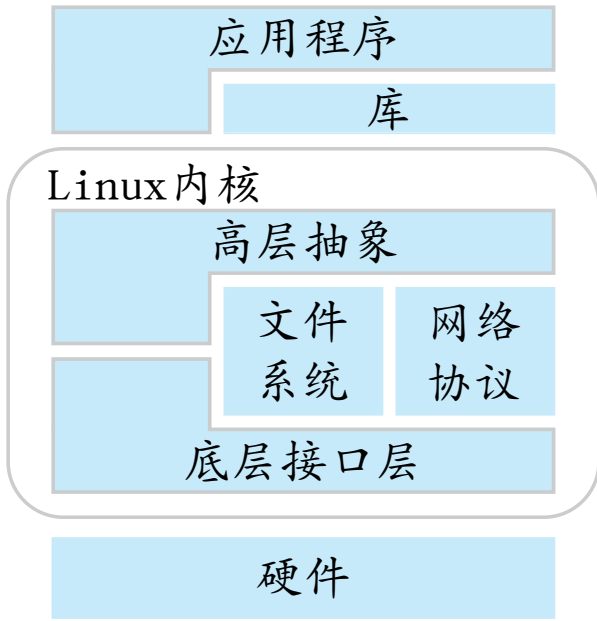
BDM，
Background Debug Mode
背景调试模式

JTAG，
Joint Test Action Group
联合测试小组
采用IEEE 1149.1，测试存取口和边界扫描标准

Outline

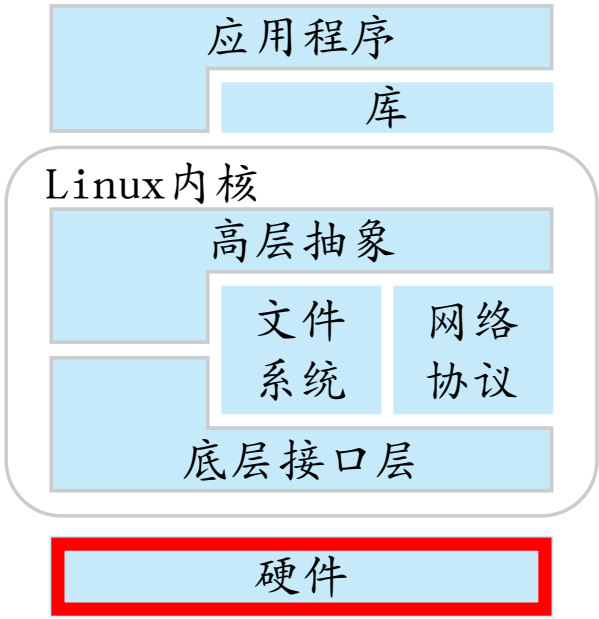
嵌入式Linux系统的一般架构

- 嵌入式Linux系统的一般架构如图所示，包含4个部分：
  1. 硬件
  2. 内核
  3. 文件系统等
  4. 应用程序/库



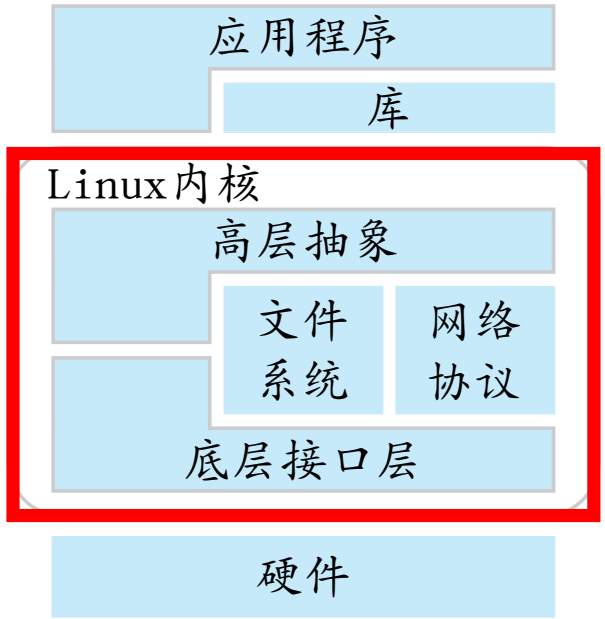
1、硬件

- 目标板的硬件必须符合一些要求方能执行Linux系统。
  - 至少32位CPU
  - 一般情况下必须配备MMU（对于不配备MMU的考虑使用uClinux）
  - RAM容量必须满足系统的需要
  - 一些最起码的I/O能力，以便在线调试
  - 具有某种形式的永久性或网络存储设备以便内核加载及（或）存取根文件系统



2、Linux内核

- Linux内核是Linux操作系统的中心组件。使用内核的目的是希望以一致的方式管理硬件，以及为用户软件提供高层抽象层。

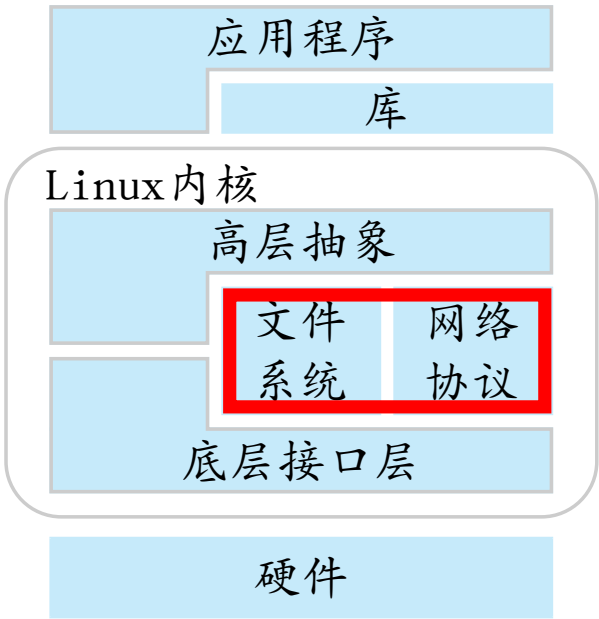


2、Linux内核

- 内核大致可以分成两个部分：
  - 底层接口层和高层抽象层
- 底层接口层专属于硬件配置，内核运行其上，并以硬件无关的高层抽象层提供对硬件资源的直接控制。
  - 比如，对于PPC和ARM系统，尽管其寄存器或内存分页的处理方式不同，但却可以使用通用的API来存取内核里高层的组件
- 通常底层部分会处理CPU特有的操作、架构特有的内存操作以及设备的基本I/O

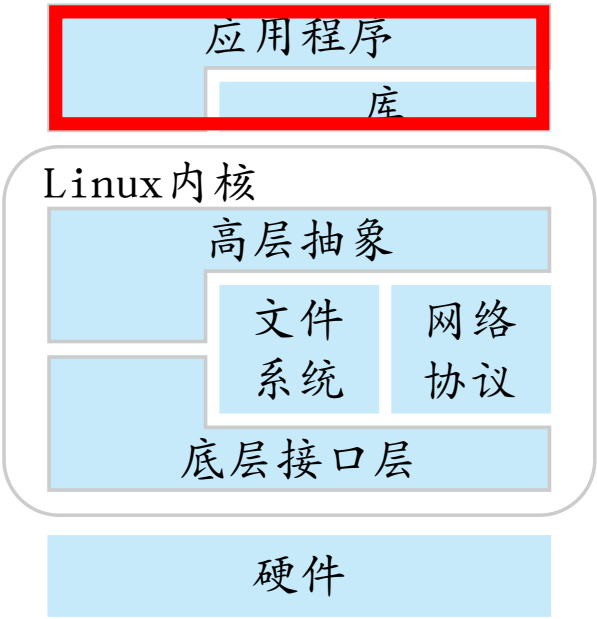
3、文件系统和网络协议等

- 在Linux内核的底层接口层与高层抽象层之间，内核有时会用到与特定设备上的结构化数据交互的组件，例如文件系统和网络协议。
- 通常，Linux内核至少需要一个具有合适结构的根文件系统。Linux内核会从中加载第一个应用程序、加载模块并为进程提供工作目录。



4、应用程序/库

- 内核上面是应用程序和工具程序。链接库通常与应用程序动态链接在一起



Outline

系统启动过程

- 在系统启动过程里，有3个主要软件组件参与其中：
  1. 引导加载程序
  2. 内核
  3. Init进程

1 引导加载程序

- 引导加载程序是系统启动过程中执行的第一个软件，它与目标板的硬件有高度的依赖关系。
- Linux有许多引导加载程序可用。
- 引导加载程序在完成底层硬件初始化工作后会接着跳到内核的启动程序代码执行。

系统启动过程

2. 内核
   - 内核一开始的启动程序代码会因架构不同而有很大的差异，而且在为C程序代码设置合适的执行环境之前，它会先为自己进行初始化工作。
   - 完成以上工作后，内核会跳到与架构无关的start_kernel函数执行，此函数会初始化高层内核功能，安装根文件系统，以及启动init进程
3. Init进程
   - 启动各种应用程序（根据设置）

Outline

引导配置的类型

- Linux系统的引导配置与所选用的引导加载程序、它的配置以及主机中软硬件的类型有非常密切的关系。
  1. 固态存储媒体
  2. 磁盘
  3. 网络

1. 固态存储媒体
   - 固态存储媒体用于存放
     - ★ 最初的引导加载程序
     - ★ 配置参数
     - ★ 内核
     - ★ 根文件系统
   - 嵌入式Linux系统在开发的不同阶段可能会使用不同的引导配置，但大部分在开发完成后使用固态存储媒体

引导配置的类型

2. 磁盘
   - 磁盘引导配置方式广泛应用于工作站及服务器中，此时内核和根文件系统位于磁盘上
   - 最初的内核加载程序不是从磁盘上加载，第二个内核加载程序就是直接从磁盘获得内核本身
   - 可以用于嵌入式系统的开发阶段
   - 要求：
     目标板上能够使用硬盘或者具有模仿硬盘的装置
3. 网络
   - 网络引导配置方式中，存在两种情况：
     1. 内核位于固态存储设备上或磁盘上，需要通过NFS安装根文件系统
     2. 只有内核加载程序位于目标板的存储设备上，需要通过TFTP下载内核和根文件系统（或NFS）
   - 往往用于开发初期

Outline

Linux的配置和编译

- 内核是所有Linux系统软件组成的核心。它的性能对整个系统的性能起决定性作用。如果内核不支持目标板上的某个硬件，那么在目标板上使用这个内核时，这个硬件就不能起作用。
- 下面讨论如何为一个嵌入式系统准备好一个可用的Linux内核，包括内核的选择、配置、编译和安装。

Linux内核源代码中的主要子目录 I

- Documentation 内核方面的相关文档。
- arch 与体系结构相关的代码。
  对应于每个支持的体系结构，有一个相应的目录，
  如i386、arm、alpha等。
  每个体系结构子目录下包含几个主要的子目录：
  - kernel 与体系结构相关的核心代码
  - mm 与体系结构相关的内存管理代码
  - lib 与体系结构相关的库代码
- include 内核头文件。
  对每种支持的体系结构有相应的子目录。
- init 内核初始化代码。
- kernel 核心代码。
- mm 内存管理代码。
- ipc 进程间通信代码。

Linux内核源代码中的主要子目录 II

- net 网络部分代码。
- lib 与体系结构无关的内核库代码。
- drivers 设备驱动代码。
  每类设备有相应的子目录，如char、block、net等
- fs 文件系统代码。
  每个支持文件系统有相应的子目录，如ext2、proc等。
- modules 可动态加载的模块。
- Scripts 配置核心的脚本文件。

Linux的配置和编译步骤

- Linux内核从配置到安装大致有如下步骤：
  - 清理：make mrproper
  - 配置：make config/menuconfig/xconfig
  - 建立依赖关系：make dep
  - 编译：make或make zImage
  - 安装：make install
- 但在嵌入式系统开发中，并不总是按照上述步骤

1、Linux内核选择

- 尽管http://www.kernel.org是主要的内核来源，但这里可用的内核版本并不总适用于嵌入式系统
- 下面列出了针对当前主要嵌入式系统的Linux内核源代码下载地点，通常直接提供了针对某种目标硬件系统的Linux内核版本

处理器系统	合适的内核站点
x86	http://www.kernel.org
ARM	http://www.arm.linux.org.uk
PowerPC	http://penguinppc.org
MIPS	http://www.linux-mips.org
M68K	http://linux-m68k.org
nonMMU的CPUs	http://www.uclinux.org

- 为了获得适合目标系统的内核，必须从主要的站点下载内核，还要打上合适的补丁。
  例如ARM Linux仅仅发布Linux官方内核的补丁。
- 找到合适的Linux内核版本后，把它下载到某个目录中，解压

2、Linux内核的配置

- 配置有很多种方法，配置过程中有很多选项可选
- 配置的结果是生成一个.config文件以及大量的符号连接和头文件，用于后续的过程。

.config文件中保存了在配置过程中定义的变量，在Linux内核目录下的Makefile中将会包含这个文件

μCLinux的配置和编译 I

- μCLinux通常以发行版的形式发布，不仅有内核，还提供根文件系统
  - 最新版本：uClinux-dist-20140504
- 考虑将μCLinux编译后运行到SkyEye平台上
  - 下载源码、建立交叉编译环境
    - ★ 阅读SOURCE文件

参考版本：uClinux-dist-20140504，下载（tar.gz或者tar.bz2）
经实际使用，可以为此版本的GDB/armulator或者skyeye的交叉编译环境为： arm-linux-tools-20061213.tar.gz

  - 配置编译

μCLinux的配置和编译 II

export LDLIBS=-ldl（可能需要）
（还可能需要安装一些软件，请根据错误信息提示安装所需软件）

make xconfig
（1）在vendor/product选项中选择GDB/ARMulator
（2）Kernel版本选择2.4.x
（3）其他选项不变（使用缺省选项）

make dep; make

- 编译完成之后
  - 编写skyeye.conf，内容如下（不带根文件系统）：

μCLinux的配置和编译 III

skyeye.conf

```
#skyeye config file sample
cpu: arm7tdmi
mach: at91
mem_bank: map=M, type=RW, addr=0x00000000, size=0x00004000
mem_bank: map=M, type=RW, addr=0x01000000, size=0x00400000
mem_bank: map=I, type=RW, addr=0xf0000000, size=0x10000000
```

- 运行：

```
skyeye -e linux -c skyeye.conf
```

由于没有根文件系统，在内核准备加载根文件系统时会报错！

ARMLinux的配置和编译

- 考虑将ARM Linux运行到SkyEye模拟平台上：
  1. 下载源代码，建立交叉编译环境
  2. 编译内核
  3. 需要自己制作根文件系统：
     利用busybox生成一个简单的系统文件并且配置根文件系统
  4. 配置SkyEye系统信息，使上述生成的内核及操作系统可以在SkyEye中运行

准备ARM Linux

- 早期：
  下载标准Linux的内核源码和ARM Linux，其中ARM Linux 是基于标准Linux内核为ARM 做的补丁
- 例如：
  - 标准Linux的内核源代码：linux-2.4.18.tar.bz2
  - ARM Linux的补丁：patch-2.4.18-rmk7.bz2
- 解压缩Linux-2.4.18

$$\text{tar -jvxf linux} - 2.4.18.\text{tar.bz2}$$

- 解压缩补丁，并对linux-2.4.18打补丁

$$\text{bzip2} - \text{d patch} - 2.4.18 - \text{rmk7.bz2}$$

$$\text{cd linux}$$

$$\text{patch} - \text{p1} < ../\text{patch} - 2.4.18.\text{rmk7}$$

- 建立armlinux-2.4.18，将linux目录拷贝到该目录下

建立交叉编译环境 I

- 下载适合arm1inux-2.4.18的交叉编译工具
  - ftp://ftp.arm.linux.org.uk/pub/arm1inux/toolchain
  - cross-2.95.3.tar.bz2
- 解压缩到/usr/local/arm目录下或当前工作目录下

$$tar\ -jxf\ cross-2.95.3.tar.bz2$$

- 设置执行路径，在.bashrc中添加交叉编译器的bin目录

$$export\ PATH = \$PATH : /usr/local/arm/2.95.3/bin$$

- 然后，退出控制台，重新启动控制台

建立交叉编译环境 II

- 检查是否建立好交叉编译环境

$$arm - linux - < tab >$$

然后

$$arm - linux - gcc \quad - v$$

arm-linux-<tab>

```
arm-linux-addr2line  arm-linux-gasp      arm-linux-protoize
arm-linux-ar         arm-linux-gcc       arm-linux-ranlib
arm-linux-as         arm-linux-gcj       arm-linux-readelf
arm-linux-c++        arm-linux-ld        arm-linux-size
arm-linux-c++filt    arm-linux-nm        arm-linux-strings
arm-linux-g++        arm-linux-objcopy   arm-linux-strip
arm-linux-g77        arm-linux-objdump   arm-linux-unprotoize
```

arm-linux-gcc -v

```
Reading specs from /usr/local/arm/2.95.3/lib/gcc-lib/arm-linux/2.95.3/specs gcc
version 2.95.3 20010315 (release)
```

配置和编译arm linux

- 修改Makefile中
  - 目标板体系结构

```
#ARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ -e s/arm.*/arm/ -e s/sa110/arm/)
ARCH := arm
```

  - 交叉编译器

```
CROSS_COMPILE = arm-linux-
```

我们考虑在skyeye上模拟ep7312 I

- 下载armlinux4skyeye，解压缩

```
tar -zvxf armlinux4skyeye-v0.2.3.tar.gz
```

- 进入armlinux4skyeye，查看文件installguide.txt，
  然后为Skyeye模拟的ep7312进行如下修改：
  1. linux-2.4.x/drivers/char/Makefile, 增加
     obj-$(CONFIG_LCD) += lcd_drv.o
     obj-$(CONFIG_TOUCH_SCREEN) += skyeye_ts_drv.o
  2. linux-2.4.x/drivers/char/Config.in, 增加
     tristate ' SkyEye LCD support(for EP7312)' CONFIG_LCD
     tristate ' SkyEye Touch Screen support(for EP7312)'
     CONFIG_TOUCH_SCREEN
  3. 增加lcd_drv.[ch], lcd_struct.h, skyeye_ts_drv.[ch],
     ep7312_sys.h files 到 armlinux-2.4.18/drivers/char/目录中

我们考虑在skyeye上模拟ep7312 II

- 清除，命令：

```
make mrproper

make clean
```

- 复制.config文件

```
cp armlinux4skyeye/example/.config armlinux-2.4.18/
```

- 配置，命令

```
make menuconfig
```

  - 在System Type中选择CLPS711X/EP721X和CLEP7312
  - 在File System中，确保
    - /proc file system support
    - ROM file system support
    - Second extended fs support
  - 在Block Devices选择

我们考虑在skyeye上模拟ep7312 III

- ★ RAM disk support
- ★ Initial RAM disk (initrd) support

▶ 在Character devices选择

- ★ SkyEye LCD support(for EP7312)
- ★ SkyEye Touch Screen support(for EP7312)

- 建立依赖关系并编译，命令

```
make dep;make
```

- 检查是否存在编译好的ARM Linux内核
- 使用skyeye运行vmlinux

配置、编译并运行linux-2.6.26

- 之前介绍了缺省编译linux-2.6.26
- 下面对linux-2.6.26进行简单的配置
  1. 从skyeye-testsuite-1.3.4_rc1/skyeye-testsuite/linux/s3c2410/cs8900a/目录中复制配置文件

     linux_config_2_6_14

     为源代码目录中的.config文件
  2. 然后

     make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
     根据需要进行配置修改（也可以不修改），保存配置

- 编译：

  make ARCH＝arm CROSS_COMPILE=arm-linux-

- 使用交叉编译器arm-linux-tools-20061213.tar.gz 可以编译成功
- 运行
  - 在skyeye-testsuite-1.3.4_rc1/skyeye-testsuite/linux/s3c2410/cs8900a/中

    skyeye -e PATH_TO/vmlinux
    （提示根文件系统有问题，后面再考虑）

Outline

Outline

根文件系统及其目录骨架 I

- Linux内核在系统启动期间进行的最后操作之一就是安装根文件系统。
- 根文件系统一直都是所有类UNIX系统不可或缺的组件
- 根文件系统的顶层目录各有其特殊的用法和目的。
  - 其中一部分往往与多用户有关
  - 在嵌入式系统中，这一部分是不必要的
- 根文件系统中的内容由FHS（Filesystem Hierarchy Standard，文件系统层次标准） 制定
  - 制定该标准的组织为FHSG（FHS Group，http://www.pathname.com/fhs/）

根文件系统及其目录骨架 II

1. 观察主机系统Kubuntu-14.04的根目录
   （或者所安装的其他发行版）

```
ls /
```

```
bin     initrd.img      media   sbin   vmlinuz
boot    initrd.img.old  mnt     srv    vmlinuz.old
cdrom   lib             opt     sys
dev     lib32           proc    tmp
etc     lib64           root    usr
home    lost+found      run     var
```

根文件系统及其目录骨架 III

2. 观察uClinux中romfs的根目录

uClinux-dist/romfs目录下的内容：

```
tree uClinux-dist/romfs/ -L 1
```

```
uClinux-dist/romfs/
├── bin
├── dev
├── etc
├── home
├── lib
├── mnt
├── proc
├── sbin
├── tmp -> /var/tmp
├── usr
└── var
```

根文件系统及其目录骨架 IV

- 若为嵌入式系统建立根文件系统，
  - 首先为多用户提供的可扩展环境的所有目录都应该省略
    - ★ /home，/mnt，/opt，/root
  - 甚至可以不要
    - ★ /tmp和/var，这要根据实际情况确定
  - 根据引导加载程序和它的配置情况，决定是否需要/boot
  - 下列几个是比较重要的
    - ★ /bin, /dev, /etc, /lib, /proc, /sbin, /usr
  - /usr和/var这两个顶层目录与根目录非常像，有自己的目录结构

容易混淆的几个目录之一

- /bin, /sbin, /usr/bin, /usr/sbin
  - 普通用户和超级用户都比较有用的命令放在/bin下
  - 普通用户不使用，只有超级用户比较有用的命令放在/sbin下
  - 不常用的用户命令放在/usr/bin下
  - 不常用的超级用户命令放在/usr/sbin下

容易混淆的几个目录之二

- /lib, /usr/lib
  - 系统启动所需要的以及上述比较有用的命令所需要的库文件通常放在/lib下
  - 所有其他的库文件一般都放在/usr/lib下，有的软件包会在/usr/lib下为自己所需的库文件建立一个专门的目录
    - ★ 例如Perl 5.x安装完后，会产生一个/usr/lib/per15目录

设置根文件系统的目录骨架

- 创建rootfs，在此目录下建立根文件系统的目录骨架。命令如下：

```
mkdir rootfs
cd rootfs
mkdir bin dev etc lib proc sbin tmp usr var
chmod 1777 tmp
mkdir usr/bin usr/lib usr/sbin
mkdir var/lib var/lock var/log var/run var/tmp
chmod 1777 var/tmp
```

设置根文件系统的目录骨架

tree rootfs

```
rootfs
├── bin
├── dev
├── etc
├── lib
├── proc
├── sbin
├── tmp
├── usr
│   ├── bin
│   ├── lib
│   └── sbin
└── var
    ├── lib
    ├── lock
    ├── log
    ├── run
    └── tmp
```

Outline

根文件系统上的内容包括：

- 链接库
- 内核模块
- 内核映像
- 设备文件
- 系统应用程序
- 系统初始化文件
- ……

根文件系统的内容：链接库

- 为目标系统准备链接库
  1. glibc
  2. uClibc

1、glibc I

- glibc套件包含若干链接库。主要包含4种类型的文件
  1. 实际的共享链接库，文件名为

     libLIBRARY_NAME-GLIBC_VERSION.so

     例如glibc-2.19的数学链接库为目录/lib/i386-linux-gnu/下的

     libm-2.19.so

  2. 主修订版本的符号链接，文件名为

     libLIBRARY_NAME.so.MAJOR_REVISION_VERSION

     例如实际的数学链接库

     libm-2.19.so

     其符号连接的名称为目录/lib/i386-linux-gnu/下的

     libm.so.6

1、glibc II

3. 与版本无关的符号链接，指向主修订版本的符号链接，
用于为需要链接特定链接库的所有程序提供一个通用的条目，
与主修订版本号或glibc涉及的版本无关。文件名为

libLiBRARY_NAME.so

例如/usr/lib/i386-linux-gnu/目录下

libm.so

指向

libm.so.6

后者指向实际的共享链接库

libm-2.19.so

1、glibc III

4. 静态链接库包文件， 文件名格式为

libLIBRARY_NAME.a

如数学库的静态包文件就是/usr/lib/i386-linux-gnu/目录下

libm.a

- 我们只需前两种。
  其余的文件只在链接执行文件时才会用到，
  执行应用程序时不需要

试使用下面的命令查看系统中有多少数学库相关的库文件

```
locate libm.
```

动态链接器及其符号连接 I

- 除了链接库文件，还需要复制动态链接器及其符号连接
- 动态链接器的文件名通常叫做

  ld-GLIBC_VERSION.so

  例如/lib/i386-linux-gnu/目录下的

  ld-2.19.so

- 动态链接器的符号链接
  - 对于i386、arm或m68k，通常为

    ld-linux.so.MAJOR_REVISION_VERSION

    例如/lib/和/lib/i386-linux-gnu/目录下的

    ld-linux.so.2

动态链接器及其符号连接 II

试使用下面的命令来查看你的系统中动态链接库相关的文件

```
locate ld-2
locate ld-linux
```

- 对于MIPS或PPC，则通常为

  ld.so. MAJOR_REVISION_VERSION

- 在向目标板的根文件系统实际复制任何 glibc组件前，应先找出应用程序需要哪 些glibc组件。
  - 可以使用ldd命令显示在主机上运行的文件所依赖的库

ldd /usr/bin/skyeye

```
linux-vdso.so.1 => (0x00007fffdf9fc000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f39250ee000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3924d28000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3925416000)
```

动态链接器及其符号连接 III

- 但是，对将要运行在目标端的命令，主机的ldd可能不行，此时最好使用交叉编译环境提供的相关命令

2、μClibc

- μClibc是glibc的替代品，实现了部分必要的链接库。
- 网站：http://www.uclibc.org/
- 若下载的是源代码，则

```
make clean
make config
make CROSS=arm-linux-
make PREFIX=<根文件系统目录> install
```

- 若下载的是已经编译好的，则需要将库文件拷贝到根文件系统目录下的lib目录中
- 若只拷贝需要的库文件，则需要采用类似glibc的方法找出目标板所依赖的uClibc
- 一般情况下，为了更好的使用uClibc，需要安装与uClibc相配套的交叉编译工具链，可以到uClibc网站获取帮助，通常需要打补丁

根文件系统的内容：内核模块和内核映像

- 为目标系统准备内核模块
  - 如果已经建立好内核模块，就将它们复制到目标板的/lib目录里
- 为目标系统准备内核映像
  - 这与引导加载程序的能力和配置有关
  - 如果设置成从根文件系统启动内核，就要将内核映像复制到目标板的根文件系统的/boot目录下

根文件系统的内容：设备文件

- 在Linux根文件系统中，所有的设备文件都放在/dev目录里，下面列出了一些基本的/dev条目

文件名	说明	类型	主设备号	次设备号	权限位
mem	物理内存存取	字符	1	1	600
null	null设备	字符	1	3	666
zero	以0值字节为数据来源	字符	1	5	666
random	随机数产生器	字符	1	8	644
tty0	当前的虚拟控制台	字符	4	0	600
tty1	第一个虚拟控制台	字符	4	1	600
ttyS0	第一个UART串行端口	字符	4	64	600
tty	当前的tty设备	字符	5	0	666
console	系统控制台	字符	5	1	600

根文件系统的内容：设备文件

- 可以使用如下的命令建立上表中的几个条目（需要root权限）

```
mknod -m 600 dev/mem c 1 1
mknod -m 666 dev/null c 1 3
mknod -m 666 dev/zero c 1 5
mknod -m 644 random c 1 8
mknod -m 600 tty0 c 4 0
mknod -m 600 tty1 c 4 1
mknod -m 600 ttyS0 c 4 64
mknod -m 666 tty c 5 0
mknod -m 600 console c 5 1
```

根文件系统的内容：设备文件

- 此外，/dev目录下还包含若干必要的符号链接，如
  - fd→/proc/self/fd
  - stdin→fd/0
  - stdout→fd/1
  - stderr→fd/2

命令为：

```
ln -s /proc/self/fd fd
ln -s fd/0 stdin
ln -s fd/1 stdout
ln -s fd/2 stderr
```

根文件系统的内容：设备文件

- 看一下：
  1. 我们建立的dev目录
  2. 虚拟机中Linux主机上的 dev目录
  3. 用过的romfs的dev目录，例如skyeye-testsuite中的

根文件系统的内容：应用程序

- Linux拥有丰富的命令，但是嵌入式Linux并不需要这么多的命令
- 有两种方法：
  - 选择少量有用的Linux命令
  - 尽可能包含多的命令，但是对命令的功能进行裁减
- 对于后者，介绍3个有用的套件
  - BusyBox
  - TinyLogin
  - Embutils

BusyBox: The Swiss Army Knife of Embedded Linux

- BusyBox目前由Denys Vlasenko来维护
  - 网站：http://www.busybox.net/
  - 下载：http://www.busybox.net/downloads/
- 它把许多常见应用程序缩微版本组合到一个单独的小巧的可执行程序中，一般含有比较少的选项，更小的体积，不过所包含的这些选项能够提供用户所需要的大部分功能。
- 能够为任何一个小型或嵌入式系统提供一个相当完整的环境
- 提供相当程度的模块化功能，很容易为目标板定制

BusyBox: The Swiss Army Knife of Embedded Linux

- 在busybox的网站上，称busybox：
  - combines tiny versions of many common UNIX utilities into a single small executable.
    - ★ 可以取代GNU fileutils, shellutils, etc.
    - ★ have fewer options than their full-featured GNU cousins
    - ★ provides a fairly complete environment for any small or embedded system
  - BusyBox has been written with size-optimization and limited resources in mind
  - 模块化、易定制
- 最新版本
  - 2014年：busybox-1.22.1
- 阅读busybox网站上的FAQ

busybox举例

- 下载busybox-1.22.1.tar.bz2
- 解压缩

```
tar jvxf busybox-1.22.1.tar.bz2
```

阅读INSTALL文件

- 使用make help可以看到完整的配置和安装选项

busybox举例

- 常规的配置和安装

```
The BusyBox build process is similar to the Linux kernel build:

make menuconfig     # This creates a file called ".config"
make                # This creates the "busybox" executable
make install        # or make CONFIG_PREFIX=/path/from/root install
```

- 简单的配置和安装，可以使用

  make defconfig

- 若在源代码外面编译，则首先建立一个新的目录，然后在那个目录中编译

```
cd ..
mkdir build_busybox
cd build_busybox
make KBUILD_SRC=../busybox-1.22.1 -f ../busybox-1.22.1/Makefile defconfig
```

若不清楚安装在什么地方，建议不要随便make install！

busybox举例

- 体系结构的选择
  - 缺省为针对i386编译
  - 若针对arm，则要指明ARCH和CROSS_COMPILE
    - ★ 经实验，针对arm，使用allnoconfig，直接编译ok
    - ★ 若使用defconfig，则会发生错误，
      需要配合menuconfig把发生错误的模块禁止

```
mkdir build_busybox_arm
cd build_busybox_arm
make KBUILD_SRC=../busybox-1.22.1 -f ../busybox-1.22.1/Makefile ARCH=arm
CROSS_COMPILE=arm-linux- allnoconfig
make ARCH=arm CROSS_COMPILE=arm-linux- busybox
```

- ★ 使用file命令看一下生成的文件
- ★ 正常使用需要根据实际情况对busybox进行配置

busybox举例

- 安装busybox
  - 对于i386平台的busybox，安装到我们指定的目录下

```
mkdir install_busybox
cd build_busybox
make CONFIG_PREFIX=../install_busybox install
```

    - ★ 使用tree命令查看install_busybox目录中的内容

busybox举例

- 运行busybox
  - 对于针对i386平台的busybox，可以运行

```
./busybox ash
```

    进入busybox的shell界面
    （看起来，与主机的shell界面好像一样）
    （使用exit命令可以返回主机shell界面）

- 对于arm的，要运行在相应的平台上

TinyLogin

- 网站：http://tinylogin.busybox.net/
- 下载：http://tinylogin.busybox.net/downloads/
- TinyLogin将许多登录工具放在单个二进制文件中，通常会与BusyBox并用，两者由相同的开发者维护
- TinyLogin中的大多数命令要使用root权限执行
- 关于busybox与tinylogin的关系

TinyLogin举例

- 下载tinylogin-1.2
- 解压缩，然后配置
- 使用glibc或者uclibc的交叉编译器对其进行编译，例如
  - 3.3.2那个版本是OK的

```
make CROSS=arm-linux- PREFIX=~/rootfs all
```

- 在root权限下将tinylogin-1.2安装到根文件系统目录中

```
sudo make PREFIX=~/rootfs install
```

Embutils

- 网站：http://www.fefe.de/embutils/
- 这是针对主流Unix命令提供的一组经过简化和优化的替代品。目前支持ARM、i386、PPC和MIPS
- 其维护者与diet libc相同，只能静态链接diet libc

定制应用程序

- 自己的应用程序也要放在根文件系统的某个目录下，这取决于应用程序所拥有的组件数量和类型
  - 如果二进制文件较少，可以考虑放到/bin目录下
  - 如果二进制文件多且复杂并且包含一些数据文件，最好在根文件系统中增加一个单独的目录，例如/project
  - 第二种情况下，通常需要设置PATH环境变量，以便能够找到可执行文件

准备系统初始化文件

- 系统初始化也是Unix系统很重要的一部分，正如我们之前所说，内核的最后一部初始化操作为启动init进程，这个程序负责创建一些其他进程并且启动系统的一些关键组件运行
  - Init可以看成是所有进程的父亲
- 在Linux中，init进程模仿了System V的init，这对于嵌入式Linux而言，功能太强大
- 我们将介绍
  - 标准的system V 初始化
  - BusyBox初始化
  - Ubuntu的初始化

关于init的进一步说明

- 事实上，内核并不关心init进程是哪一个，init进程只不过代表了内核在初始化完成后要启动应用程序
- 我们可以修改启动参数让内核使用我们自己的init

$$\text{init} = \text{PATH\_TO\_YOUR\_INIT}$$

  - 缺点在于，这样只能启动我们自己的应用，如果有必要还需要承担标准init的一部分工作，例如启动其他必要的系统组件
  - 更进一步，当我们的程序出现异常时，可能导致整个系统的关闭或者重启。在有的情况下，这就是系统所希望的，但在大多数情况下，这样做是无用的。
  - 因此，比较安全的方法就是使用一个真正的init程序

标准的system V 初始化 I

- 标准的init包在多数Linux发行版本中都能找到，也可以在

  ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/

  上找到，目前由Miquel van Soorenburg维护
- 包含的命令有：
  - halt，init，killall5，last，mesg，runlevel，shutdown，sulogin，utmpdump，以及wall
- 下载到源码之后，首先解压缩，然后使用交叉编译器编译

  make CC=arm-linux-gcc
- 安装到根文件系统中

  make BIN_OWNER=“$(id —un)” BIN_GROUP=“$(id —gn)” > ROOT=根文件系统目录 install

标准的system V 初始化 II

- 由于我们使用当前用户权限，而Makefile默认使用root权限，因此可能会失败，这可以忽略。因为目标系统中不考虑多用户。否则可以在root权限下做。
- 若使用root权限，要小心设置ROOT指向目标系统的根文件系统，否则将覆盖主机上的相应程序。由于目标码不同，这将导致系统出错。
- 安装完init程序后，需要增加/etc/inittab文件，并在/etc/rc.d目录中增加一些文件
  - /etc/inittab定义runlevels
  - /etc/rc.d目录定义各个runlevels上运行的服务

7个运行级别

运行级别	说明
0	系统处于halt状态
1	只有一个用户，无需login
2	多用户，无NFS，命令行形式的login
3	完整的多用户模式，命令行形式的login
4	未使用
5	x11，图形界面形式的login
6	系统reboot

- 在大多数主机上，缺省的runlevel为5
- 在嵌入式系统上，可以设置为1，此时没有访问控制
- 系统启动之后，我们仍然可以修改runlevel，这就需要在新老init进程之间使用FIFO进行通信
- 因此需要创建一个FIFO，命令为：

```
mknod -m 600 ROOT_PATH/dev/initctl p
```

BusyBox初始化

- BusyBox也提供类似init的功能，适合用于嵌入式系统
- BusyBox不提供runlevel功能
- 在我们前面安装的BusyBox中，sbin/init是/bin/busybox的符号链接，因此BusyBox是系统启动后运行的第一个应用程序
- BusyBox将调用它的init

BusyBox的init I

- Init主要执行下列任务
  1. 初始化init的信号处理函数
  2. 初始化console控制台
  3. 解释/etc/inittab文件
  4. 运行系统初始化脚本，BusyBox缺省使用 /etc/init.d/rcS
  5. 运行所有inittab的阻塞式命令
  6. 运行所有inittab中的一次性执行命令
- 完成上述任务之后，init就进入一个死循环，在这个死循环中执行下列任务
  1. 运行所有必须再生的命令
  2. 运行所有必须被请求才能响应的命令
- 在BusyBox初始化console控制台的时候，根据系统的配置进行初始化
  - 如，在启动参数中console=ttyS0，表示使用串口

BusyBox的init II

- 在初始化完console之后，busybox将会检查是否存在/etc/inittab，如果没有将会使用缺省的inittab配置
  - 参见busybox-1.22.1/examples/inittab
- 缺省的inittab设置，如
  - 系统重启，系统停止，init重启
  - 还有，在最先的4个虚拟console：tty1~tty4上启动shell

Inittab文件的格式

- Inittab文件中每一行有下列格式

  id:runlevel:action:process

- 在busybox中，
  - id代表tty的序号
  - 忽略runlevel
  - Process说明要运行的程序的路径和命令选项
  - Action说明process的执行方式

8种执行方式 I

1. sysinit：提供init的路径
2. respawn：每当一个命令结束后，就重启该命令
3. askfirst：类似respawn，但是要先问一下用户
4. wait：阻塞式命令，init要等待其运行完毕
5. once：只运行一次，不必等待
6. ctrlaltdel：三键齐按时，要执行的命令
7. shutdown：系统关闭时执行
8. restart：系统重启时执行，通常就是init

8种执行方式 II

- 一个可能的inittab如下 （id和runlevel都为空）

```
::sysinit:/etc/init.d/rcS
::respawn:/sbin/getty 115200 ttyS0
::respawn:/control-module/bin/init
::restart:/sbin/init
::shutdown:/bin/umount -a -r
```

设置/etc/init.d/rcS作为系统初始化文件
在串口（115200波特率）启动一个登录会话
启动控制模块定制的系统初始化脚本
设置/sbin/init为重启时运行的命令
系统关闭时，运行umount

系统初始化脚本 I

- 根据系统初始化脚本的设置不同，其功能可以很强大
- 通常
  - 重新挂载根文件系统，以可读可写
  - 挂载其他文件系统
  - 初始化并启动网络
  - 启动系统守护进程

举例

```
#!/bin/sh
# Remount the root filesystem in read-write (requires /etc/fstab)
mount -n -o remount,rw /
# Mount /proc filesystem
mount /proc
# Start the network interface
/sbin/ifconfig eth0 192.168.172.10
```

/etc/fstab举例

```
# /etc/fstab
# device directory type options
#
/dev/nfs / nfs defaults
none /proc proc defaults
```

制作根文件系统

- 准备好根文件系统的内容后，就要设置可供目标板使用的根文件系统
  - 选择根文件系统的类型
  - 制作根文件系统的映像或安装根文件系统到目标设备上

Outline

选择根文件系统的依据

- 描绘一个嵌入式文件系统的特性通常包括：
  - 可被写入：
    这个文件系统可被写入么？
  - 具有永久性：
    重引导后，这个文件系统可以保存修改过的内容么？
  - 具有断电可靠性：
    经变动的文件系统可以在断电之后恢复过来么？
  - 经过压缩：
    经安装的文件系统，其内容经过压缩么？
  - 存在RAM中：
    文件系统的内容在被安装之前会先从存储设备取出并放到RAM中么？

选择根文件系统的依据

- 下表列出了常见的几种嵌入式文件系统及其特性

文件系统	可被写入	具有永久性	具有断电可靠性	经过压缩	存在于RAM中
CRAMFS	否	不适用	不适用	是	否
JFFS2	是	是	是	是	否
JFFS	是	是	是	否	否
NFTL上的Ext2	是	是	否	否	否
NFTL上的Ext3	是	是	是	否	否
RAM disk上的Ext2	是	否	否	否	是

CRAMFS

- 这是Linux Torvalds编写的只具备最基本特性的文件系统，它非常简单、经过压缩并且只读，主要用于嵌入式系统，具有以下限制：
  - 每个文件最大不超过16MB
  - 不提供当前目录“.”和上级目录“..”
  - 文件的UID字段只有16位，GID字段只有8位
  - 所有文件的时间戳为Unix epoch（00:00:00 GMT, January 1, 1970）
  - 内存分页大小必须是4096
  - 文件链接计数器永远是1
- 要为根文件系统建立CRAMFS映像，首先要建立并安装CRAMFS工具：cramfsck和mkcramfs
- 可以在内核源代码树的scripts/cramfs目录里找到他们的程序代码，在该目录下使用make就可以建立这两个工具
- 使用mkcramfs命令建立CRAMFS映像：
  - mkcramfs 根文件系统的根目录 映像名

RAMdisk I

- 存在于RAM中，其存取功能类似于块设备
- 内核可以在同一时间支持多个活动的RAMdisk
- 在RAMdisk上可以使用任何磁盘文件系统
- RAMdisk通常会从经压缩的磁盘文件系统（如ext2）加载其内容，因此内核必须具备从存储设备取出initrd（initial RAM disk）映像
  作为它的根文件系统的能力。
- 启动时，内核会确认引导选项是否指示有initrd的存在，如果有就会从所选定的存储设备取出文件系统映像放入RAM disk，并且将它安装成根文件系统

RAMdisk II

- 看一下skyeye中ep7312/2.6.x下的initrd.img中有些什么

```
sudo mount -o loop initrd.img rootfs/
tree rootfs -L 1
```

```
rootfs
├── bin
├── dev
├── etc
├── home
├── lib
├── linuxrc -> bin/busybox
├── lost+found
├── proc
├── root
├── sbin
├── tmp
├── usr
└── var

12 directories, 1 file
```

建立供RAM disk使用的文件系统映像 I

1. 创建一个新的mount点

   ```
   mkdir initrd
   ```

2. 以dd命令建立一个8192KB的文件系统映像

```
dd if=/dev/zero of=initrd_new.img bs=1k count=8192
8192+0 records in
8192+0 records out
8388608 bytes (8.4 MB) copied, 0.497361 s, 16.9 MB/s
```

此时，initrd_new.img全0，相当于一块空白的硬盘

建立供RAM disk使用的文件系统映像 II

3. 在建立好的文件系统映像上建立文件系统

```
mke2fs -F -v -m0 initrd_new.img
```

-F选项指定对mke2fs在文件上运行而不是块设备
-v选项以verbose模式运行
-m0指出不必为超级用户保留任何区块

建立供RAM disk使用的文件系统映像 III

mke2fs -F -v -m0 initrd_new.img

```
mke2fs 1.42.9 (4-Feb-2014)
fs_types for mke2fs.conf resolution: 'ext2', 'small'
Discarding device blocks: done
Discard succeeded and will return 0s - skipping inode table wipe
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
2048 inodes, 8192 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=8388608
1 block group
8192 blocks per group, 8192 fragments per group 2
048 inodes per group

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

建立供RAM disk使用的文件系统映像 IV

4. 接下来，就可以将initrd_new.img挂载到刚刚建立的mount点上

   sudo mount -o loop initrd_new.img initrd/

   - 当然，此时文件系统中基本上是空的

```
tree initrd/
```

```
initrd/
└── lost+found [error opening dir]

1 directory, 0 files
```

5. 复制根文件系统到RAM disk
   - 可以是我们自己制作的
   - 也可以是之前挂载的ep7312的initrd.img

建立供RAM disk使用的文件系统映像 V

```
sudo cp -av rootfs2/* initrd/
```

tree initrd/ -L 1

```
initrd/
├── bin
├── dev
├── etc
├── home
├── lib
├── linuxrc -> bin/busybox
├── lost+found
├── proc
├── root
├── sbin
├── tmp
├── usr
└── var

12 directories, 1 file
```

建立供RAM disk使用的文件系统映像 VI

6. 卸载根文件系统

   ```
   sudo umount initrd/
   ```

7. 现在initrd_new.img文件中已经包含了目标板的整个根文件系统
8. 最后形成经压缩的RAM disk

   ```
   gzip -9 < initrd_new.img >initrd_new.bin
   ```

   -9表示最高级的压缩算法

```
gzip -9 < initrd_new.img >initrd_new.bin
file initrd_new.bin
```

```
initrd_new.bin: gzip compressed data, last modified: Sat Nov 29 21:02:28 2014, max
compression, from Unix
```

使用skyeye-testsuite下的根文件系统运行uClinux

1. 使用apt-get安装的skyeye
   - 在skyeye-testsuite-1.3.4_rc1/skyeye-testsuite/uClinux/at91/uclinux_cs8900a目录下，直接运行

     ```
     /usr/bin/skyeye -e YOUR_PATH/uClinux-dist/linux-2.4.x/linux
     ```

使用skyeye-testsuite下的根文件系统运行uClinux

2. 使用编译的skyeye-1.3.5
   - 在skyeye-testsuite-1.3.4_rc1/skyeye-testsuite/uClinux/at91/uclinux_cs8900a目录下，修改skyeye.conf如下：

skyeye.conf

```
skyeye config file sample
arch:arm
cpu: arm7tdmi
mach: at91
mem_bank: map=M, type=RW, addr=0x00000000, size=0x00004000
mem_bank: map=M, type=RW, addr=0x01000000, size=0x00400000
mem_bank: map=M, type=R, addr=0x01400000, size=0x00400000, file=./romfs.img
mem_bank: map=M, type=RW, addr=0x02000000, size=0x00400000
mem_bank: map=M, type=RW, addr=0x02400000, size=0x00008000
mem_bank: map=M, type=RW, addr=0x04000000, size=0x00400000
mem_bank: map=I, type=RW, addr=0xf0000000, size=0x10000000
#set nic info #net: type=cs8900a, base=0xfffa0000, size=0x20,int=16, mac=0:4:3:2:1:f,
ethmod=tuntap, hostip=10.0.0.1
net: type=cs8900a, ethmod=tuntap, hostip=10.0.0.1
uart: mod = term
#dbct: state=on
```

使用skyeye-testsuite下的根文件系统运行uClinux

2. 使用编译的skyeye-1.3.5
   - 然后运行：

     skyeye -e YOUR_PATH/uClinux-dist/linux-2.4.x/linux

     或者

     PATH/skyeye -e YOUR_PATH/uClinux-dist/linux-2.4.x/linux

     ★ 进入skyeye交互界面后，运行start和run

```
SkyEye 1.3.5
SkyEye is an Open Source project under GPL.
...
Get more information about it, please visit the homepage http://www.skyeye.org.
Type ”help” to get command list.
(skyeye) start
...
(skyeye) run
(running) ...
```

★ start之后会出现一个模拟器窗口

★ run之后，在窗口中会显示uClinux启动信息，最后进入uClinux交互界面

Outline

作业

- 在嵌入式Linux系统开发中，存在哪几种主机/目标机开发体系结构？
- 主机/目标板的调试方式有哪几种？
- 嵌入式Linux系统的引导配置的类型有哪几种？

Thanks !

The end.