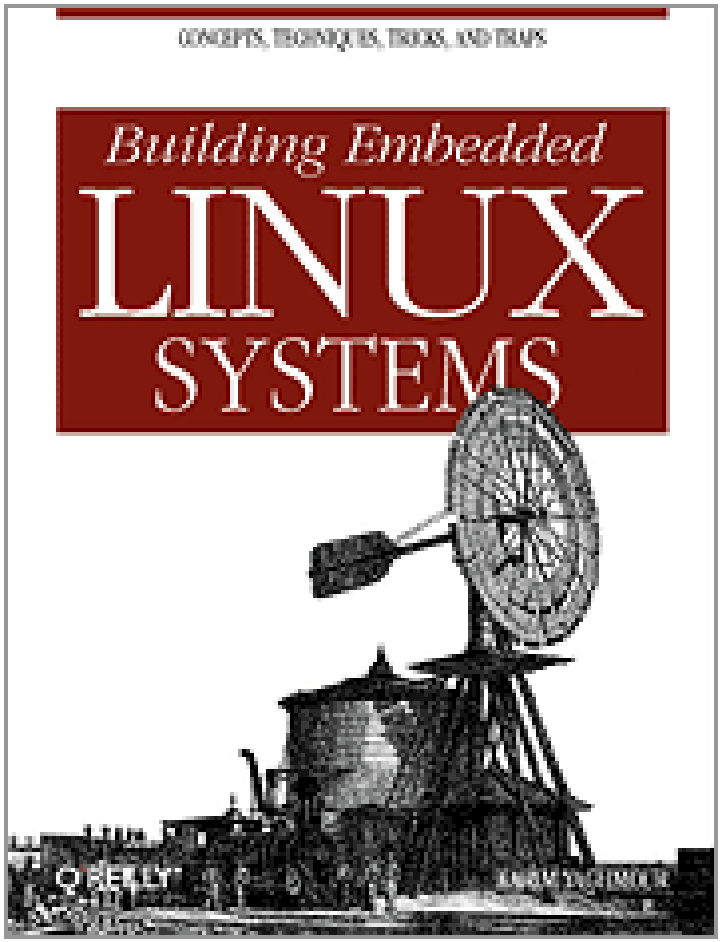
嵌入式操作系统

Building Embedded Linux Systems
构建嵌入式 Linux 系统
之
根文件系统及其制作

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

/media/SAMSUNG/work/6 实验室相关


CONCEPTS, TECHNIQUES, TRICKS, AND TRAPS
Building Embedded
LINUX
SYSTEMS
O'REILLY

Outline

Outline

根文件系统的目录架构 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

根文件系统的目录架构 II

- 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/）
- 查看 Linux 下的根目录（本机）
- 查看 romfs 下的目录

建立根文件系统

- 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

- **/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
- **/lib, /usr/lib**
  - 系统启动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/perl5 目录

设置根文件系统的目录骨架

```
xlanchen@xlanchen-desktop:~$ mkdir rootfs
xlanchen@xlanchen-desktop:~$ cd rootfs
xlanchen@xlanchen-desktop:~/rootfs$ mkdir bin dev etc lib proc sbin tmp usr var
xlanchen@xlanchen-desktop:~/rootfs$ chmod 1777 tmp
xlanchen@xlanchen-desktop:~/rootfs$ ls
bin  dev  etc  lib  proc  sbin  tmp  usr  var
xlanchen@xlanchen-desktop:~/rootfs$ mkdir usr/bin usr/lib usr/sbin
xlanchen@xlanchen-desktop:~/rootfs$ mkdir var/lib var/lock var/lob var/run var/tmp
xlanchen@xlanchen-desktop:~/rootfs$ chmod 1777 var/tmp
xlanchen@xlanchen-desktop:~/rootfs$ ls usr
bin  lib  sbin
xlanchen@xlanchen-desktop:~/rootfs$ ls var
lib  lob  lock  run  tmp
xlanchen@xlanchen-desktop:~/rootfs$
```

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
  - ……

Outline

为目标系统准备链接库

1. glibc
2. uClibc

glibc I

- glibc 套件包含若干链接库。
- 主要包含 4 种类型的文件
- 实际的共享链接库，文件名为 libLIBRARY_NAME-GLIBC_VERSION.so
  - 例如 glibc 2.5 的数学链接库为 libm-2.5.so

```
xlanchen@xlanchen-desktop:~$ ls /lib/lib*.so
/lib/libanl-2.9.so            /lib/libmemusage.so         /lib/libpcprofile.so
/lib/libBrokenLocale-2.9.so   /lib/libnsl-2.9.so          /lib/libproc-3.2.7.so
/lib/libc-2.9.so              /lib/libnss_compat-2.9.so   /lib/libpthread-2.9.so
/lib/libcidn-2.9.so           /lib/libnss_dns-2.9.so      /lib/libresolv-2.9.so
/lib/libcrypt-2.9.so          /lib/libnss_files-2.9.so    /lib/librt-2.9.so
/lib/libdl-2.9.so             /lib/libnss_hesiod-2.9.so   /lib/libSegFault.so
/lib/libkeyutils-1.2.so       /lib/libnss_nis-2.9.so      /lib/libthread_db-1.0.so
/lib/libm-2.9.so              /lib/libnss_nisplus-2.9.so  /lib/libutil-2.9.so
xlanchen@xlanchen-desktop:~$
```

glibc II

- 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-2.5.so，其符号连接的名称为 libm.so.6

```
xlanchen@xlanchen-desktop:~$ ls /lib/libm* -la
-rw-r--r-- 1 root root 149328 2009-09-01 02:46 /lib/libm-2.9.so
-rw-r--r-- 1 root root  13740 2009-09-01 02:46 /lib/libmemusage.so
lrwxrwxrwx 1 root root     11 2009-09-11 20:56 /lib/libm.so.6 -> libm-2.9.so
xlanchen@xlanchen-desktop:~$
```

glibc III

- 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 libm.so 指向 libm.so.6，libm.so.6 指向实际的共享链接库 libm-2.2.3.so

```
xlanchen@xlanchen-desktop:~$ ls /usr/lib/libm.so -la
lrwxrwxrwx 1 root root 14 2009-09-11 20:56 /usr/lib/libm.so -> /lib/libm.so.6
xlanchen@xlanchen-desktop:~$
```

glibc IV

- 静态链接库包文件，文件名格式为 libLIBRARY_NAME.a，如动态装载库 libdl 的静态包文件就是 libdl.a

```
xlanchen@xlanchen-desktop:~$ ls /usr/lib/*.a
/usr/lib/libanl.a           /usr/lib/libgio-2.0.a      /usr
/usr/lib/libasprintf.a      /usr/lib/libglib-2.0.a     /usr
/usr/lib/libaudio.a         /usr/lib/libGLU.a          /usr
/usr/lib/libBrokenLocale.a  /usr/lib/libgmodule-2.0.a  /usr
/usr/lib/libbsd-compat.a    /usr/lib/libgobject-2.0.a  /usr
/usr/lib/libc.a             /usr/lib/libgthread-2.0.a  /usr
/usr/lib/libchmfile.a       /usr/lib/libICE.a          /usr
/usr/lib/libc_nonshared.a   /usr/lib/libieee.a         /usr
/usr/lib/libcom_err.a       /usr/lib/libjpeg.a         /usr
/usr/lib/libcrypt.a         /usr/lib/libkdeextra.a     /usr
/usr/lib/libcrypto.a        /usr/lib/liblcms.a         /usr
/usr/lib/libcurses.a        /usr/lib/libm.a            /usr
/usr/lib/libdl.a            /usr/lib/libmcheck.a       /usr
/usr/lib/libexpat.a         /usr/lib/libmenu.a         /usr
/usr/lib/libexpatw.a        /usr/lib/libmng.a          /usr
/usr/lib/libfontconfig.a    /usr/lib/libncurses.a      /usr
/usr/lib/libform.a          /usr/lib/libncurses++.a    /usr
/usr/lib/libfreetype.a      /usr/lib/libnsl.a          /usr
/usr/lib/libg.a             /usr/lib/libpanel.a        /usr
/usr/lib/libgettextpo.a     /usr/lib/libpgport.a       /usr
xlanchen@xlanchen-desktop:~$
```

glibc V

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

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

```
xlanchen@xlanchen-desktop:~$ ls /lib/ld-* -la
-rwxr-xr-x 1 root root 117348 2009-09-01 02:46 /lib/ld-2.9.so
lrwxrwxrwx 1 root root      9 2009-09-11 20:56 /lib/ld-linux.so.2 -> ld-2.9.so
xlanchen@xlanchen-desktop:~$
```

- 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
  - 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

```
xlanchen@xlanchen-desktop:~$ ldd /bin/mkdir
        linux-gate.so.1 =>  (0xb7f4c000)
        libselinux.so.1 => /lib/libselinux.so.1 (0xb7f1b000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7db8000)
        libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7db3000)
        /lib/ld-linux.so.2 (0xb7f4d000)
xlanchen@xlanchen-desktop:~$
```

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

- 例如 arm-linux-readelf -d busybox

```
xlanchen@xlanchen-desktop:~/workspace/make_romfs/busybox-1.9.2$
/usr/local/arm/3.3.2/bin/arm-linux-readelf -d busybox

Dynamic segment at offset 0xca850 contains 22 entries:
  标记        类型                         名称/值
 0x00000001 (NEEDED)                     共享库：[libcrypt.so.1]
 0x00000001 (NEEDED)                     共享库：[libm.so.6]
 0x00000001 (NEEDED)                     共享库：[libc.so.6]
 0x0000000c (INIT)                       0xd358
 0x0000000d (FINI)                       0xb51bc
```

uClibc I

- uClibc 是 glibc 的替代品，实现了部分必要的链接库。
- 网站：http://www.uclibc.org/
- uClibc - a Small C Library for Linux
  - a C library for developing embedded Linux systems.
  - much smaller than the GNU C Library, but nearly all applications supported by glibc also work perfectly with uClibc.
  - Porting applications from glibc to uClibc typically involves just recompiling the source code.
  - It currently runs on standard Linux and MMU-less (also known as µClinux) systems with support for alpha, ARM, cris, e1, h8300, i386, i960, m68k, microblaze, mips/mipsel, PowerPC, SH, SPARC, and v850 processors.

uClibc II

- 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

Outline

准备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

Outline

为目标系统建立设备文件 I

- 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

为目标系统建立设备文件 II

- 可以使用如下的命令建立上表中的几个条目（需要 root 权限）

```
xlanchen@xlanchen-desktop:~/rootfs/dev$ ls
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 600 mem c 1 1
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 666 null c 1 3
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 666 zero c 1 5
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 666 random c 1 8
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 600 tty0 c 4 0
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 600 tty1 c 4 1
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 600 ttyS0 c 4 64
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 666 tty c 5 0
xlanchen@xlanchen-desktop:~/rootfs/dev$ sudo mknod -m 600 console c 5 1
xlanchen@xlanchen-desktop:~/rootfs/dev$ ls
console  mem  null  random  tty  tty0  tty1  ttyS0  zero
xlanchen@xlanchen-desktop:~/rootfs/dev$
```

为目标系统建立设备文件 III

- 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

```
xlanchen@xlanchen-desktop:~/rootfs/dev$ ln -s /proc/self/fd fd
xlanchen@xlanchen-desktop:~/rootfs/dev$ ln -s fd/0 stdin
xlanchen@xlanchen-desktop:~/rootfs/dev$ ln -s fd/1 stdout
xlanchen@xlanchen-desktop:~/rootfs/dev$ ln -s fd/2 stderr
xlanchen@xlanchen-desktop:~/rootfs/dev$ ls
console  mem   random  stdin   tty   tty1   zero
fd       null  stderr  stdout  tty0  ttyS0
xlanchen@xlanchen-desktop:~/rootfs/dev$
```

为目标系统建立设备文件 IV

```
xlanchen@xlanchen-desktop:~/rootfs/dev$ ls -la
总用量 8
drwxr-xr-x  2 xlanchen xlanchen  4096 2009-10-12 17:35 .
drwxr-xr-x 11 xlanchen xlanchen  4096 2009-10-12 16:42 ..
crw-------  1 root     root     5,  1 2009-10-12 17:32 console
lrwxrwxrwx  1 xlanchen xlanchen    13 2009-10-12 17:35 fd -> /proc/self/fd
crw-------  1 root     root     1,  1 2009-10-12 17:31 mem
crw-rw-rw-  1 root     root     1,  3 2009-10-12 17:31 null
crw-rw-rw-  1 root     root     1,  8 2009-10-12 17:31 random
lrwxrwxrwx  1 xlanchen xlanchen     4 2009-10-12 17:35 stderr -> fd/2
lrwxrwxrwx  1 xlanchen xlanchen     4 2009-10-12 17:35 stdin -> fd/0
lrwxrwxrwx  1 xlanchen xlanchen     4 2009-10-12 17:35 stdout -> fd/1
crw-rw-rw-  1 root     root     5,  0 2009-10-12 17:32 tty
crw-------  1 root     root     4,  0 2009-10-12 17:31 tty0
crw-------  1 root     root     4,  1 2009-10-12 17:31 tty1
crw-------  1 root     root     4, 64 2009-10-12 17:31 ttyS0
crw-rw-rw-  1 root     root     1,  5 2009-10-12 17:31 zero
xlanchen@xlanchen-desktop:~/rootfs/dev$
```

为目标系统建立设备文件 V

- 下图为用过的 romfs 的 dev 目录，对照 Linux 主机上的 dev 目录

```
xlanchen@xlanchen-desktop:~/workspace/skyeye-testsuite-1.2.8/uClinux/at91/uclinux_cs
8900a$ ls romfs/dev/
console  ptyp0  ptyp6  ptypc  random  rom5  tty0  ttyp2  ttyp8  ttype
cua0     ptyp1  ptyp7  ptypd  rom0    rom6  tty1  ttyp3  ttyp9  ttypf
cua1     ptyp2  ptyp8  ptype  rom1    rom7  tty2  ttyp4  ttypa  ttyS0
kmem     ptyp3  ptyp9  ptypf  rom2    rom8  tty3  ttyp5  ttypb  ttyS1
mem      ptyp4  ptypa  ram0   rom3    rom9  ttyp0  ttyp6  ttypc  urandom
null     ptyp5  ptypb  ram1   rom4    tty   ttyp1  ttyp7  ttypd  zero
xlanchen@xlanchen-desktop:~/workspace/skyeye-testsuite-1.2.8/uClinux/at91/uclinux_cs
8900a$
```

Outline

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（已经与 busybox 合并）
  - Embutils

BusyBox: The Swiss Army Knife of Embedded Linux I

- 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
- 在 busybox 的网站上，称 busybox：
  - combines tiny versions of many common UNIX utilities into a single small executable.

BusyBox: The Swiss Army Knife of Embedded Linux II

- 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
- 阅读 busybox 网站上的 FAQ

例如 I

- 下载 busybox 源码包，并解压缩
- 阅读 INSTALL 文件
- 使用 make help 可以看到完整的配置和安装选项
- 常规的配置和安装

```
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

```
make defconfig
make
```

make install

- 缺省为针对 i386 编译

例如 II

- 若针对 arm，则要指明 ARCH 和 CROSS_COMPILE
  - 经实验，针对 arm，使用 allnoconfig，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
- 对于针对 i386 平台的 busybox，可以运行
  - ./busybox ash
    进入 busybox 的 shell 界面
- 对于 arm 的，要运行在相应的平台上
- 实例，参见本文件后面的
  - busybox-1.00 的静态编译和安装

例如 III

- busybox-1.9.2 的静态编译和安装
- busybox-1.9.2 的动态编译和安装

TinyLogin I

- 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 was merged into BusyBox, current sources can thus be checked out via BusyBox.
  - In the busybox source tree, just
  - make allnoconfig && make menuconfig
  - and select the appropriate applets from the “Login/Password Management Utilities” menu.

TinyLogin II

- 例（历史，仅仅用做参考）
  - 下载 tinylogin-1.2
  - 解压缩，然后配置
  - 使用 glibc 或者 uclibc 的交叉编译器 (3.3.2 OK) 对其进行编译，例如

```
donger@donger:~/tinylogin-1.2$ 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

TinyLogin III

```
donger@donger:~/tinylogin-1.2$ sudo make PREFIX=~/rootfs install
Password:
  /home/donger/rootfs/bin/addgroup -> tinylogin
  /home/donger/rootfs/bin/adduser -> tinylogin
  /home/donger/rootfs/bin/delgroup -> tinylogin
  /home/donger/rootfs/bin/deluser -> tinylogin
  /home/donger/rootfs/bin/login -> tinylogin
  /home/donger/rootfs/bin/su -> tinylogin
  /home/donger/rootfs/bin/tinylogin -> tinylogin
  /home/donger/rootfs/sbin/getty -> ../bin/tinylogin
  /home/donger/rootfs/sbin/sulogin -> ../bin/tinylogin
  /home/donger/rootfs/usr/bin/passwd -> ../../bin/tinylogin
  /home/donger/rootfs/usr/bin/vlock -> ../../bin/tinylogin
donger@donger:~/tinylogin-1.2$
```

TinyLogin IV

```
donger@donger:~/rootfs$ ls bin
addgroup  chown     dmesg   ln      mv      sh          true
adduser   cp        echo    login   pidof   sleep       umount
ash       date      false   ls      ps      su          uname
busybox   dd        grep    mkdir   pwd     sync        zcat
cat       delgroup  gunzip  mknod   rm      tar
chgrp     deluser   gzip    more    rmdir   tinylogin
chmod     df        kill    mount   sed     touch
donger@donger:~/rootfs$ ls sbin
getty  init   lsmod   modprobe  reboot   swapoff  syslogd
halt   klogd  mkswap  poweroff  sulogin  swapon
donger@donger:~/rootfs$ ls usr/bin
[         cut       find  killall  sort  uniq    which
basename  dirname   free  logger   tail  uptime  whoami
chvt      du        head  passwd   test  vlock   xargs
clear     env       id    reset    tty   wc      yes
donger@donger:~/rootfs$ ls usr/sbin
chroot
donger@donger:~/rootfs$
```

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

Outline

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

Outline

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
- 可以修改启动参数让内核使用我们自己的 init
  - init=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
- 下载源码，解压缩，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**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 I

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

7 个运行级别 II

- 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
  - mknod — m 600 根文件系统目录 /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

BusyBox 的 init II

- 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
- 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
- 缺省的 inittab 设置，如
  - 系统重启，系统停止，init 重启
  - 以及在最先的 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

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

- 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

- busybox 中使用 /etc/init.d/rcS
- 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

- 每个文件系统（包括分区或者设备）用一行来描述
- 在每一行中，用空格或 TAB 符号来分隔各个字段
  设备挂载点文件系统类型选项
  device directory type options
- 举例
  ```
  # /etc/fstab
  # device directory type options
  #
  /dev/nfs / nfs defaults
  none /proc proc defaults
  ```

Outline

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

选择根文件系统的依据 I

- 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？
- 下表列出了常见的几种嵌入式文件系统及其特性

选择根文件系统的依据 II

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 I

- 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

CRAMFS II

- 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

- 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（例如 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
- 看一下 skyeye 中 ep7312/ep7312_with_imagekernel 下的 initrd.img 中有些什么

建立供 RAM disk 使用的文件系统映像 I

- 创建一个新的 mount 点，initrd
  - 命令：**mkdir initrd**
- 以**dd 命令**建立一个 8192KB 的文件系统映像，并以 /dev/zero 对它进行初始化

```
xlanchen@xlanchen-desktop:~$ mkdir initrd
xlanchen@xlanchen-desktop:~$ dd if=/dev/zero of=initrd.img bs=1k count=8192
记录了8192+0 的读入
记录了8192+0 的写出
8388608字节(8.4 MB)已复制，0.0348422 秒，241 MB/秒
xlanchen@xlanchen-desktop:~$
```

建立供 RAM disk 使用的文件系统映像 II

- 在文件系统映像上建立文件系统（需要 root 权限）

mke2fs initrd.img

```
xlanchen@xlanchen-desktop:~$ mke2fs initrd.img
mke2fs 1.41.4 (27-Jan-2009)
initrd.img is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2048 inodes, 8192 blocks
409 blocks (4.99%) reserved for the super user
First data block=1
Maximum filesystem blocks=8388608
1 block group
8192 blocks per group, 8192 fragments per group
2048 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 26 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
xlanchen@xlanchen-desktop:~$
```

建立供 RAM disk 使用的文件系统映像 III

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

```
xlanchen@xlanchen-desktop:~$ sudo mount -o loop initrd.img initrd
[sudo] password for xlanchen:
xlanchen@xlanchen-desktop:~$ ls initrd
lost+found
xlanchen@xlanchen-desktop:~$
```

  - 可以看到，此时文件系统中基本上是空的
- 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
- 卸载根文件系统
  - 命令：sudo umount initrd
- 现在 initrd.img 文件中已经包含了目标板的整个根文件系统

建立供 RAM disk 使用的文件系统映像 IV

- [可选]最后形成经压缩的 RAM disk
  - 命令：gzip -9 < initrd.img > initrd.bin

Outline

阅读 μCLinux2008 中的源代码，了解 romfs 的生成 I

- 阅读文件 uClinux-dist/vendors/GDB/ARMulator/Makefile
  - 了解目录的创建

```
7 ROMFS_DIRS = bin dev etc home lib mnt proc sbin usr var
```

阅读 μCLinux2008 中的源代码，了解 romfs 的生成 II

- 了解 dev 目录下设备文件的创建

```
 9 DEVICES = \
10 »       tty,c,5,0       console,c,5,1       cua0,c,5,64       cua1,c,5,65  \
11 »       \
12 »       mem,c,1,1       kmem,c,1,2          null,c,1,3        ram0,b,1,0 \
13 »       ram1,b,1,1 \
14 »       \
15 »       ptyp0,c,2,0     ptyp1,c,2,1         ptyp2,c,2,2       ptyp3,c,2,3 \
16 »       ptyp4,c,2,4     ptyp5,c,2,5         ptyp6,c,2,6       ptyp7,c,2,7 \
17 »       ptyp8,c,2,8     ptyp9,c,2,9         ptypa,c,2,10      ptypb,c,2,11 \
18 »       ptypc,c,2,12    ptypd,c,2,13        ptype,c,2,14      ptypf,c,2,15 \
19 »       \
20 »       rom0,b,31,0     rom1,b,31,1         rom2,b,31,2       rom3,b,31,3 \
21 »       rom4,b,31,4     rom5,b,31,5         rom6,b,31,6       rom7,b,31,7 \
22 »       rom8,b,31,8     rom9,b,31,9 \
23 »       \
24 »       tty0,c,4,0      tty1,c,4,1          tty2,c,4,2        tty3,c,4,3 \
25 »       ttyS0,c,4,64    ttyS1,c,4,65 \
26 »       \
27 »       ttyp0,c,3,0     ttyp1,c,3,1         ttyp2,c,3,2       ttyp3,c,3,3 \
28 »       ttyp4,c,3,4     ttyp5,c,3,5         ttyp6,c,3,6       ttyp7,c,3,7 \
29 »       ttyp8,c,3,8     ttyp9,c,3,9         ttypa,c,3,10      ttypb,c,3,11 \
30 »       ttypc,c,3,12    ttypd,c,3,13        ttype,c,3,14      ttypf,c,3,15 \
31 »       \
32 »       zero,c,1,5      random,c,1,8        urandom,c,1,9
```

阅读 μCLinux2008 中的源代码，了解 romfs 的生成 III

```
37 romfs:
38 »       [ -d $(ROMFSDIR)/$$i ] || mkdir -p $(ROMFSDIR)
39 »       for i in $(ROMFS_DIRS); do \
40 »       »       [ -d $(ROMFSDIR)/$$i ] || mkdir -p $(ROMFSDIR)/$$i; \
41 »       done
42 »       for i in $(DEVICES); do \
43 »       »       touch $(ROMFSDIR)/dev/@$$i; \
44 »       done
45 »       # these permissions are needed for openpty and family to work
46 »       # on non-ptmx ptys
47 »       chmod 620 $(ROMFSDIR)/dev/@[pt]ty[pqrsPQRS][0-9a-f],*
48 »       $(ROMFSINST) -s /var/tmp /tmp
49 »       #$(ROMFSINST) -s /bin /sbin
50 »       $(ROMFSINST) /etc/rc
51 »       $(ROMFSINST) /etc/inittab
52 »       $(ROMFSINST) ../../Generic/romfs/etc/services /etc/services
53 »       case "$(LINUXDIR)" in \
54 »       *2.4.*) ;; \
55 »       *2.6.*) ;; \
56 »       *) echo "ttyS0:linux:/bin/sh" >> $(ROMFSDIR)/etc/inittab ;; \
57 »       esac
58 »       $(ROMFSINST) /etc/motd
59 »       $(ROMFSINST) /etc/passwd
60 »       echo "$(VERSIONSTR) -- " `date` > $(ROMFSDIR)/etc/version
```

阅读 μCLinux2008 中的源代码，了解 romfs 的生成 IV

```
62 image:
63 »       [ -d $(IMAGEDIR) ] || mkdir -p $(IMAGEDIR)
64 »       genromfs -v -V "ROMdisk" -f $(ROMFSIMG) -d $(ROMFSDIR)
65 »       cp $(ROOTDIR)/$(LINUXDIR)/linux $(IMAGEDIR)
```

Outline

编译 busybox I

- 下载：http://www.busybox.net/downloads/
- 最新版本：busybox-1.15.1.tar.bz2，解压缩
- 阅读 INSTALL 文件
- 运行

  make help

- 运行如下命令：

  make defconfig
  make

编译 busybox II

思考：

上述缺省编译的 busybox 能在什么平台上运行？

- 运行如下命令：

```
./busybox ash
ls
...
```

基于 busybox 制作根文件系统 (arm-linux 版) I

- 对于 initrd.img，首先准备根文件系统的空白映像
  dd if=/dev/zero of=initrd.img bs=1k count=2048
- 将空白的 init.img 格式化成 ext2 文件系统
  mkfs.ext2 initrd.img
- 将空白的根文件系统挂载到 rootfs 目录中
  mkdir rootfs
  sudo mount -o loop initrd.img rootfs
- 准备根文件系统的目录框架
  cd rootfs
  sudo mkdir root home sbin etc dev usr lib tmp mnt sys proc
  sudo mkdir usr/lib usr/bin

基于 busybox 制作根文件系统 (arm-linux 版) II

- 准备根文件系统所需的文件
  - shell：使用 busybox（假设 busybox 已经编译好）
    cd **/path/to/busybox**
    make ... CONFIG_PREFIX=**/path/to/**rootfs install
  - 库
    - 根据需要
  - 启动脚本文件
    cp -rf **/path/to/busybox**/examples/bootfloppy/etc/*
    /path/to/rootfs/etc
  - 设备文件
    cd **/path/to/**rootfs/dev
    sudo mknod -m 660 console c 5 1
    sudo mknod -m 660 null c 1 3

基于 busybox 制作根文件系统 (arm-linux 版) III

- 卸载 rootfs
  sudo umount rootfs
- 从 skyeye-testsuite-1.2.8/linux/s3c2410/s3c2410x-2.6.14/ 目录下复制 skyeye.conf 文件和 vmlinux 文件
- 运行 skyeye 命令
  skyeye -c skyeye.conf -e vmlinux

Outline

busybox-1.00 的编译 I

- 交叉编译器：**arm-linux-gcc-3.3.2.tar.bz2**
  - 在主机根目录下，运行：
    sudo tar jvxf /FILE_DIR/arm-linux-gcc-3.3.2.tar.bz2
    - 此时，安装好的交叉编译器在 /usr/local/arm/3.3.2 目录下
- busybox 版本：**busybox-1.00.tar.bz2**
- 下载后，在工作目录下解压缩，后运行如下命令
  cd busybox-1.00
  make defconfig
  make menuconfig
- 修改配置如下：

busybox-1.00 的编译 II

- Build Options —>
  [*] Build BusyBox as a static binary (no shared libs)
  [ ] Build with Large File Support (for accessing files > 2 GB)
  [*] Do you want to build BusyBox with a Cross Compiler?
  (**/usr/local/arm/3.3.2/arm-linux/bin/**) Cross Compiler prefix
  () Any extra CFLAGS options for the compiler?
- Installation Options —>
  [*] Don't use /usr
  (./_install) BusyBox installation prefix
  - NetWorking Utilities：取消 **route**

- 然后运行
  make dep
  make busybox

busybox-1.00 的安装

make install

- 查看_install 目录下的内容
- 若已知安装目录 DIR，则

  make PREFIX=DIR install

Outline

busybox-1.9.2 的编译 I

- 交叉编译器：**arm-linux-gcc-3.3.2.tar.bz2**
  - 在主机根目录下，运行：
    sudo tar jvxf /FILE_DIR/arm-linux-gcc-3.3.2.tar.bz2
    - 此时，安装好的交叉编译器在 /usr/local/arm/3.3.2 目录下
- busybox 版本：**busybox-1.9.2.tar.bz2**
- 下载后，在任意工作目录下解压缩，后运行如下命令

```
cd busybox-1.9.2/
make help
make ARCH=arm CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ distclean
make ARCH=arm CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ defconfig
make ARCH=arm CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ menuconfig
```

busybox-1.9.2 的编译 II

- 修改配置如下

Busybox Settings —>

- Build Options —>
  [*] Build BusyBox as a static binary (no shared libs)
- Installation Options —>
  [*] Don't use /usr

- 编译 busybox

```
make ARCH=arm CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ busybox
```

busybox-1.9.2 的编译 III

- 遇到错误

```
  CC      applets/applets.o
applets/applets.c:15:2: warning: #warning Static linking against glibc produces buggy executables
applets/applets.c:16:2: warning: #warning (glibc does not cope well with ld --gc-sections).
applets/applets.c:17:2: warning: #warning See sources.redhat.com/bugzilla/show_bug.cgi?id=3400
applets/applets.c:18:2: warning: #warning Note that glibc is unsuitable for static linking anyway.
applets/applets.c:19:2: warning: #warning If you still want to do it, remove -Wl,--gc-sections
applets/applets.c:20:2: warning: #warning from scripts/trylink and remove this warning.
applets/applets.c:21:2: #error Aborting compilation.
make[1]: *** [applets/applets.o] 错误 1
make: *** [applets] 错误 2
```

  - 打开文件 scripts/trylink，删除所有包含 **-Wl,–gc-sections** 的行
  - 并修改文件 applets/applets.c，将上述报错的几行删除
- 继续编译（make ... busybox）
- 遇到 route 出错，make ... menuconfig 中修改配置如下，
  - Networking Utilities —> 取消 **route** 的选择
- 继续编译（make ... busybox）

busybox-1.9.2 的安装

- make ... install
  - 查看_install 目录下的内容
- 若已知安装目录 DIR，则
  make ... CONFIG_PREFIX PREFIX=DIR install

Outline

busybox-1.10.1 的静态编译 I

- 交叉编译器同上
- 编译命令：

```
make ARCH=arm
CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ distclean
make ARCH=arm
CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/
defconfig
make ARCH=arm
CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/
menuconfig
```

busybox-1.10.1 的静态编译 II

- 修改配置

Busybox Settings —>

- Build Options —>

  [*] Build BusyBox as a static binary (no shared libs)

- Installation Options —>

  [*] Don't use /usr

- make ... busybox
  - 在编译过程中，遇到出错，则回到 make ... menuconfig，取消相关配置
  - 可能是：taskset、brctl、ifenslave 和 route

busybox-1.10.1 的安装

- make ... CONFIG_PREFIX=ROOTFS_DIR install

Outline

busybox-1.9.2 的动态编译 I

- 交叉编译器同上
- make ... distclean
- make ... defconfig
- make ... menuconfig
- 修改配置如下
  - Installation Options —>
    [*] Don't use /usr
- make ... busybox
  - 还是遇到 route.c 出错，仍然取消网络配置中的 route 选项
- make ... CONFIG_PREFIX=/path/to/rootfs install

busybox-1.9.2 的动态编译 II

- 查看所依赖的动态库
  - /usr/local/arm/3.3.2/bin/arm-linux-readelf -d busybox
  - 从 /usr/local/arm/3.3.2/lib/ 下将需要的文件拷贝到 rootfs/lib 目录下
    - 注意：有些文件是符号链接，对于符号链接还需要对应的文件拷贝过来
  - 此外，还需要拷贝动态链接器 ld-*

小结

1. 根文件系统的内容及其准备
   - 根文件系统的目录架构及其建立
   - 为目标系统准备链接库
   - 准备内核模块和内核映像
   - 为目标系统建立设备文件
   - 应用程序
   - 定制应用程序
   - 准备系统初始化文件
2. 根文件系统类型的选择
   - 根文件系统的类型及制作
3. 根文件系统的制作
   - 阅读 μCLinux 中的源代码，了解 romfs 的生成
   - 基于 busybox 制作根文件系统
   - busybox-1.00 的静态编译和安装
   - busybox-1.9.2 的静态编译和安装
   - busybox-1.10.1 的静态编译和安装
   - busybox-1.9.2 的动态编译和安装
4. 小结和作业

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？
- 根文件系统中的内容有哪些？

制作 initrd.img 的一个参考脚本文件 I

- 前提
  - 脚本文件和 busybox-1.00（或者 busybox-1.9.2）在同一个目录下
  - 生成的 initrd.img 也在同一个目录下
  - 使用 rootfs 作为根文件系统目录（最好原来目录下没有 rootfs 目录）
- 脚本内容如下 ()

制作 initrd.img 的一个参考脚本文件 II

```
IMG=initrd.img

VER1=1.00
VER2=1.9.2
BUSYBOX_VERSION=${VER1}

CUR_DIR=‘pwd‘
ROOTFS_DIR=rootfs
[ -d ${ROOTFS_DIR} ] && rm -rf ${ROOTFS_DIR}
mkdir ${ROOTFS_DIR}

[ -f ${IMG} ] && rm ${IMG}
dd if=/dev/zero of=${IMG} bs=1k count=2048
mkfs.ext2 ${IMG}
sudo mount -o loop ${IMG} ${ROOTFS_DIR}

cd ${ROOTFS_DIR}
sudo mkdir root home sbin etc dev usr lib tmp mnt sys proc
sudo mkdir usr/lib usr/bin

cd ../busybox-${BUSYBOX_VERSION}
if [ ${BUSYBOX_VERSION} = ${VER2} ]; then
make ARCH=arm CROSS_COMPILE=/usr/local/arm/3.3.2/arm-linux/bin/ CONFIG_PREFIX=../rootfs install
fi
```

制作 initrd.img 的一个参考脚本文件 III

```
if [ ${BUSYBOX_VERSION} = ${VER1} ]; then
sudo make PREFIX=../rootfs install
fi

sudo cp -rf ./examples/bootfloppy/etc/* ../rootfs/etc
cd ${CUR_DIR}/${ROOTFS_DIR}
sudo chmod 4755 bin/busybox

cd dev
sudo mknod -m 660 console c 5 1
sudo mknod -m 660 null c 1 3

cd ${CUR_DIR}
sudo umount ${ROOTFS_DIR}
```

Thanks !

The end.