# Linux简易根文件系统

## 1.BusyBox

BusyBox是一个集成了一百多个最常用linux命令和工具的软件。BusyBox 包含了一些简单的工具，例如ls、cat和echo等等，还包含了一些更大、更复杂的工具，例grep、find、mount以及telnet。

### 1.1 源码目录



### 1.2 基本用法

lrwxrwxrwx 1 jay jay 11 10月 27 11**:**05 linuxrc **->** bin**/**busybox

lrwxrwxrwx 1 jay jay 7 10月 27 11**:**05 ps **->** busybox

lrwxrwxrwx 1 jay jay 7 10月 27 11**:**05 pwd **->** busybox

lrwxrwxrwx 1 jay jay 7 10月 27 11**:**05 rm **->** busybox

从其构造可以看出，所有的命令都是指向busybox的软链接。/bin/busybox是BusyBox的实体。作为参数，/bin/busybox的argv[0]被设置为各个命令的名字。比如，当执行/bin/cat的时候，实际执行的是/bin/busybox。这里busybox的argv[0]为/bin/cat。

uboot传递进来的init命令为linuxrc，在busybox中查到如下代码：

USE\_FEATURE\_INITRD**(**APPLET\_NOUSAGE**(**linuxrc**,** init**,** \_BB\_DIR\_ROOT**,** \_BB\_SUID\_NEVER**))**

# define APPLET\_NOUSAGE(name,main,l,s) int main##\_main(int argc, char \*\*argv);

展开后

APPLET\_NOUSAGE**(**linuxrc**,** init**,** \_BB\_DIR\_ROOT**,** \_BB\_SUID\_NEVER**))**

**-->**init\_main

也就是说内容引导的linuxrc可执行文件的入口函数为init\_main

### 1.3 init\_main

(1)为init设置信号处理函数

(2)初始化控制台

(3)剖析/etc/inittab文件

(4) 执行系统初始化命令行，缺省情况下会使用/etc/init.d/rcS

(5)执行所有导致init暂停的inittab命令（动作类型：wait）

(6)执行所有仅执行一次的inittab（动作类型：once）

一旦完成以上工作，init进程便会循环执行以下进程：

(1)执行所有终止时必须重新启动的inittab命令(动作类型：once）

(2)执行所有终止时必须重新启动但启动前必须询问用户的inittab命令（动作类型：askfirst)

#### 1.3.1 设置信号处理函数

设置信号处理函数，例如reboot/poweroff命令就是向这里发送SIGUSR1,SIGUSR2,

SIGTERM等命令，由这里的信号处理函数进行reboot系统调用

signal**(**SIGHUP**,** exec\_signal**);**

signal**(**SIGQUIT**,** exec\_signal**);**

signal**(**SIGUSR1**,** shutdown\_signal**);**

signal**(**SIGUSR2**,** shutdown\_signal**);**

signal**(**SIGINT**,** ctrlaltdel\_signal**);**

signal**(**SIGTERM**,** shutdown\_signal**);**

signal**(**SIGCONT**,** cont\_handler**);**

signal**(**SIGSTOP**,** stop\_handler**);**

signal**(**SIGTSTP**,** stop\_handler**);**

#### 1.3.2 剖析/etc/inittab文件

busybox的inittab文件与通常的inittab不同，它没有runlevel的概念，语句功能上也有限制。  
inittab语句的标准格式是**<id>:<runlevels>:<action>:<command>。**各字段的含义如下：  
**<id>**: id字段与通常的inittab中的含义不同，它代表的是这个语句中process执行所在的tty设备，内容就是/dev目录中tty设备的文件名。由于是运行process的tty设备的文件名，所以也不能象通常的inittab那样要求每条语句id的值唯一。   
**<runlevels>**: 此字段完全被忽略。   
**<action>**: 为下列这些值之一：   
sysinit, respawn, askfirst, wait,once, restart, ctrlaltdel, shutdown   
其含义与通常的inittab的定义相同。特别提一下askfirst，它的含义与respawn相同，只是在运行process前，会打出一句话 “please press Enter to active this console”，然后等用户在终端上敲入回车键后才运行process。   
**<command>**： 指定要运行process的命令行。

对应的action示意图如下：



在这里我们的inittab文件内容如下：

# /etc/inittab

**::**sysinit**:/etc/init.**d**/**rcS

s3c2410\_serial0**::**askfirst**:-/bin/sh**

**::**ctrlaltdel**:/sbin/reboot**

**::**shutdown**:/bin/umount** **-**a **-**r

开机会运行/etc/init.d/rcS脚本,/bin/sh shell命令,关机会运行shutdown命令。  
开发板上通过mdev生成的/dev目录中,S3C2410的串口名是s3c2410\_serial0。

通过parse\_inittab函数对inittab文件进行解析，存放到init\_action\_list列表中，供后面使用。

static void parse\_inittab**(**void**)**

**{**

FILE **\***file**;**

char buf**[**INIT\_BUFFS\_SIZE**],** lineAsRead**[**INIT\_BUFFS\_SIZE**];**

char tmpConsole**[**CONSOLE\_NAME\_SIZE**];**

char **\***id**,** **\***runlev**,** **\***action**,** **\***command**,** **\***eol**;**

const struct init\_action\_type **\***a **=** actions**;**

file **=** fopen**(**INITTAB**,** "r"**);**

**if** **(**file **==** **NULL)** **{**

/\* No inittab file -- set up some default behavior \*/

/\* Reboot on Ctrl-Alt-Del \*/

new\_init\_action**(**CTRLALTDEL**,** "reboot"**,** ""**);**

/\* Umount all filesystems on halt/reboot \*/

new\_init\_action**(**SHUTDOWN**,** "umount -a -r"**,** ""**);**

/\* Swapoff on halt/reboot \*/

**if** **(**ENABLE\_SWAPONOFF**)** new\_init\_action**(**SHUTDOWN**,** "swapoff -a"**,** ""**);**

/\* Prepare to restart init when a HUP is received \*/

new\_init\_action**(**RESTART**,** "init"**,** ""**);**

/\* Askfirst shell on tty1-4 \*/

new\_init\_action**(**ASKFIRST**,** bb\_default\_login\_shell**,** ""**);**

new\_init\_action**(**ASKFIRST**,** bb\_default\_login\_shell**,** VC\_2**);**

new\_init\_action**(**ASKFIRST**,** bb\_default\_login\_shell**,** VC\_3**);**

new\_init\_action**(**ASKFIRST**,** bb\_default\_login\_shell**,** VC\_4**);**

/\* sysinit \*/

new\_init\_action**(**SYSINIT**,** INIT\_SCRIPT**,** ""**);**

**return;**

**}**

**while** **(**fgets**(**buf**,** INIT\_BUFFS\_SIZE**,** file**)** **!=** **NULL)** **{**

//跳过每一行前面的空格

**for** **(**id **=** buf**;** **\***id **==** ' ' **||** **\***id **==** '\t'**;** id**++);**

//跳过comment注释行

**if** **(\***id **==** '#' **||** **\***id **==** '\n'**)**

**continue;**

//处理尾部的\n,以字符串的形式

eol **=** strrchr**(**id**,** '\n'**);**

**if** **(**eol **!=** **NULL)**

**\***eol **=** '\0'**;**

//复制一份该行的内容

strcpy**(**lineAsRead**,** buf**);**

//解析runlev

runlev **=** strchr**(**id**,** ':'**);**

**if** **(**runlev **==** **NULL** **||** **\*(**runlev **+** 1**)** **==** '\0'**)** **{**

message**(**L\_LOG **|** L\_CONSOLE**,** "Bad inittab entry: %s"**,** lineAsRead**);**

**continue;**

**}** **else** **{**

**\***runlev **=** '\0'**;**

**++**runlev**;**

**}**

//解析action

action **=** strchr**(**runlev**,** ':'**);**

**if** **(**action **==** **NULL** **||** **\*(**action **+** 1**)** **==** '\0'**)** **{**

message**(**L\_LOG **|** L\_CONSOLE**,** "Bad inittab entry: %s"**,** lineAsRead**);**

**continue;**

**}** **else** **{**

**\***action **=** '\0'**;**

**++**action**;**

**}**

//解析command

command **=** strchr**(**action**,** ':'**);**

**if** **(**command **==** **NULL** **||** **\*(**command **+** 1**)** **==** '\0'**)** **{**

message**(**L\_LOG **|** L\_CONSOLE**,** "Bad inittab entry: %s"**,** lineAsRead**);**

**continue;**

**}** **else** **{**

**\***command **=** '\0'**;**

**++**command**;**

**}**

//放到init\_action\_list链表中

**for** **(**a **=** actions**;** a**->**name **!=** 0**;** a**++)** **{**

**if** **(**strcmp**(**a**->**name**,** action**)** **==** 0**)** **{**

**if** **(\***id **!=** '\0'**)** **{**

**if** **(**strncmp**(**id**,** "/dev/"**,** 5**)** **==** 0**)**

id **+=** 5**;**

strcpy**(**tmpConsole**,** "/dev/"**);**

safe\_strncpy**(**tmpConsole **+** 5**,** id**,**

**sizeof(**tmpConsole**)** **-** 5**);**

id **=** tmpConsole**;**

**}**

new\_init\_action**(**a**->**action**,** command**,** id**);**

**break;**

**}**

**}**

**if** **(**a**->**name **==** 0**)** **{**

/\* Choke on an unknown action \*/

message**(**L\_LOG **|** L\_CONSOLE**,** "Bad inittab entry: %s"**,** lineAsRead**);**

**}**

**}**

fclose**(**file**);**

**}**

#### 1.3.3 执行命令

init\_main中会调用run\_actions执行init\_action\_list列表中的command。

static void run\_actions**(**int action**)**

**{**

struct init\_action **\***a**,** **\***tmp**;**

**for** **(**a **=** init\_action\_list**;** a**;** a **=** tmp**)** **{**

tmp **=** a**->**next**;**

**if** **(**a**->**action **==** action**)** **{**

/\* a->terminal of "" means "init's console" \*/

**if** **(**a**->**terminal**[**0**]** **&&** access**(**a**->**terminal**,** R\_OK **|** W\_OK**))** **{**

delete\_init\_action**(**a**);**

**}** **else** **if** **(**a**->**action **&** **(**SYSINIT **|** WAIT **|** CTRLALTDEL **|** SHUTDOWN **|** RESTART**))** **{**

waitfor**(**a**,** 0**);**

delete\_init\_action**(**a**);**

**}** **else** **if** **(**a**->**action **&** ONCE**)** **{**

run**(**a**);**

delete\_init\_action**(**a**);**

**}** **else** **if** **(**a**->**action **&** **(**RESPAWN **|** ASKFIRST**))** **{**

/\* Only run stuff with pid==0. If they have

\* a pid, that means it is still running \*/

**if** **(**a**->**pid **==** 0**)** **{**

a**->**pid **=** run**(**a**);**

**}**

**}**

**}**

**}**

**}**

注意两点：  
(1)在run函数里面有调用fork，也就是，run\_actions会把每个command都放到一个新的progress里面去执行。  
(2)RESPAWN和ASKFIRST是不会从action\_list里面删除掉的！！！所以，标记为这两个ACTION的操作，在run之前会检查进程在不在，不在的话就重启，在的话什么都不做，继续执行下一跳。

主循环中处理函数如下：  
(1)按SYSINIT,WAIT,ONCE,做好init的准备工作。具体比如run\_actions(SYSINIT)。  
(2)开始一个死循环。执行RESPAWN和ASKFIRST，如果这些process不存在，就重启他们；监视子进程是否有挂掉的，如果有，就把他们的尸体拾掇拾掇，然后登记在册。

run\_actions**(**SYSINIT**);**

run\_actions**(**WAIT**);**

run\_actions**(**ONCE**);**

**while** **(**1**)**

**{**

run\_actions**(**RESPAWN**);**

run\_actions**(**ASKFIRST**);**

sleep**(**1**);**

//等待子进程挂掉

wpid **=** wait**(NULL);**

**while** **(**wpid **>** 0**)** **{**

//寻找谁挂掉了，将pid设置为0，如果是RESPAWN或ASKFIRST里面的，重启

**for** **(**a **=** init\_action\_list**;** a**;** a **=** a**->**next**)** **{**

**if** **(**a**->**pid **==** wpid**)** **{**

a**->**pid **=** 0**;**

message**(**L\_LOG**,** "process '%s' (pid %d) exited. "

"Scheduling it for restart."**,**

a**->**command**,** wpid**);**

**}**

**}**

/\* see if anyone else is waiting to be reaped \*/

wpid **=** waitpid**(-**1**,** **NULL,** WNOHANG**);**

**}**

## 2.构建根文件系统

### 2.1 bin目录

bin目录存放了busybox及各种指向busybox的链接文件。拷贝的时候使用cp –d,  
-d:若源文件为连接文件杜属性,则复制连接文件属性而非文件本身。

bin目录下的部分内容：

lrwxrwxrwx 1 jay jay 7 1月 22 2008 addgroup **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 adduser **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 ash **->** busybox

**-**rwxr**-**xr**-**x 1 jay jay 753728 1月 22 2008 busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 cat **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 catv **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 chattr **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 chgrp **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 chmod **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 chown **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 cp **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 cpio **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 date **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 dd **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 delgroup **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 deluser **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 df **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 dmesg **->** busybox

lrwxrwxrwx 1 jay jay 7 1月 22 2008 dumpkmap **->** busybox

### 2.2 sbin目录

和bin目录类似，存放指向busybox的各种链接文件。

lrwxrwxrwx 1 jay jay 14 1月 22 2008 adjtimex **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 arp **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 fdisk **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 fsck **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 fsck**.**minix **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 getty **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 halt **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 hdparm **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 hwclock **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 ifconfig **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 ifdown **->** **../**bin**/**busybox

lrwxrwxrwx 1 jay jay 14 1月 22 2008 ifup **->** **../**bin**/**busybox

### 3.3 etc目录

etc目录下主要有fstab,inittab,init.d/rcS三个文件。

inittab文件内容前面有分析过。

#### 3.3.1 /etc/fstab

fstab文件内容如下：

# device mount-point type options dump fsck order

proc **/**proc proc defaults 0 0

tmpfs **/**tmp tmpfs defaults 0 0

sysfs **/**sys sysfs defaults 0 0

tmpfs **/**dev tmpfs defaults 0 0

mount –a命令会读取fstab这个文件，进行文件系统的挂载。  
参数解析：  
(1)device：要挂载的设备。  
例如/dev/hda2,/dev/mtdblock1等设备文件，也可以是其他格式，对于proc文件系统这个字段没有意义，对于nfs文件系统，这个字段为<host>:<dir>。

(2)mount-point：挂载点。

(3)type：文件系统类型。  
例如：proc,jffs2,yaffs,ext2,nfs等，也可以是auto，表示自动检测文件系统类型

(4)options：挂载参数，以逗号隔开。

|  |  |
| --- | --- |
| Async/sync | 设置是否为同步方式运行，默认为async |
| auto/noauto | 当下载mount -a 的命令时，此文件系统是否被主动挂载。默认为auto |
| rw/ro | 是否以以只读或者读写模式挂载 |
| exec/noexec | 限制此文件系统内是否能够进行"执行"的操作 |
| user/nouser | 是否允许用户使用mount命令挂载 |
| suid/nosuid | 是否允许SUID的存在 |
| Usrquota | 启动文件系统支持磁盘配额模式 |
| Grpquota | 启动文件系统对群组磁盘配额模式的支持 |
| Defaults | 同事具有rw,suid,dev,exec,auto,nouser,async等默认参数的设置 |

(5)dump：能否被dump备份命令作用  
dump是一个用来作为备份的命令。通常这个参数的值为0或者1。

|  |  |
| --- | --- |
| 0 | 代表不要做dump备份 |
| 1 | 代表要每天进行dump的操作 |
| 2 | 代表不定日期的进行dump操作 |

(6)fsck：是否检验扇区。  
开机的过程中，系统默认会以fsck检验我们系统是否为完整（clean）。

|  |  |
| --- | --- |
| 0 | 不要检验 |
| 1 | 最早检验（一般根目录会选择） |
| 2 | 1级别检验完成之后进行检验 |

#### 3.3.2 /ect/init.d/rcS

开机执行脚本。

#!/bin/sh

//设置ip地址

ifconfig eth0 192.168.1.17

//读取/etc/fstab内容，挂载文件系统

mount **-**a

//devpts内核虚拟文件

//和proc一样。

//Linux提供给管理员通过文件系统和内核进行沟通（读\写）的一种渠道。

//pts是远程虚拟终端。devpts即远程虚拟终端文件设备。

//通过/dev/pts可以了解目前远程虚拟终端的基本情况。

//devpts文件系统加载在 /dev/pts

mkdir **/**dev**/**pts

mount **-**t devpts devpts **/**dev**/**pts

//mdev 是busybox 自带的一个简化版的udev，适合于嵌入式的应用埸合。

//其具有使用简单的特点。

//它的作用，就是在系统启动和热插拔或动态加载驱动程序时，

//自动产生驱动程序所需的节点文件。

//设置系统的hotplug程序为mdev

echo **/**sbin**/**mdev **>** **/**proc**/**sys**/**kernel**/**hotplug

//自动创建设备节点

mdev **-**s

### 3.3.3 lib目录

系统所需的动态库，这里从交叉编译工具的lib下拷贝过来  
gcc-3.4.5-glibc-2.3.6/arm-linux/lib

可以使用如下命令查询二进制程序所依赖的lib

arm**-**linux**-**readelf –a “your binary” **|** grep “Shared”

### 3.3.4 dev目录

在制作根文件系统时，需要手动创建一些设备节点。

crw**-**r**--**r**--** 1 root root 5**,** 1 1月 22 2008 console

crw**-**r**--**r**--** 1 root root 1**,** 3 1月 22 2008 null

创建命令如下：

sudo mknod null c 1 3

sudo mknod console c 5 1

### 3.3.5 usr目录

busybox对应的usr目录，提供一些命令

### 3.3.6 其他空目录

还需创建一些空目录mnt proc sys tmp root。

## 3. 文件系统制作

mkyaffsimage fs\_mini fs\_mini.yaffs

mk.jffs2 –n –s 2048 –e 128KB –f fs\_mini –o fs\_mini.jffs2

## 4. shell概念

Linux用户启动一个进程的通用方法是在shell中执行命令，命令中包括可执行程序的路径以及启动所需参数。新启动的进程是shell进程的子进程，shell使用wait系列函数等待用户进程的结束，在进程结束后wait函数会返回，从而shell收到通知并回收资源。本文主要说明shell如何启动用户进程，Linux系统中可执行文件格式ELF以及通过execve系统调用启动用户进程的过程。

**shell工作方式：**

通过shell运行命令执行一个用户进程的方法是，通过fork()创建一个子进程，在子进程中调用execve(pathname, argv, envp)加载新程序，为新进程建立文本段，创建栈、数据段以及堆，在shell进程中执行wait调用等待子进程返回。C程序代码框架大致如下：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main**(**int argc**,** char **\*** argv**[])**

**{**

    int pid**;**

    /\* fork another process \*/

    pid **=** fork**();**

    if **(**pid**<**0**)**

**{**

        /\* error occurred \*/

        exit**(-**1**);**

**}**

    else **if** **(**pid**==**0**)**

**{**

        execve**(**pathname**,** argv**,** envp**);**

**}**

    else

**{**

        wait**(NULL);**

        exit**(**0**);**

**}**

**}**