1. uboot启动内核
2. 内核：启动应用程序 挂接根文件系统
3. 构建根文件系统

在分析内核启动的时候我们知道init\_post()函数启动应用程序：

然后是如下的一些东西：

sys\_open((const char \_\_user \*) "/dev/console", O\_RDWR, 0) < 0））//打开终端对应我们的是串口

(void) sys\_dup(0);

(void) sys\_dup(0);

static int \_\_init init\_setup(char \*str)

{

unsigned int i;

execute\_command = str;

/\*

\* In case LILO is going to boot us with default command line,

\* it prepends "auto" before the whole cmdline which makes

\* the shell think it should execute a script with such name.

\* So we ignore all arguments entered \_before\_ init=... [MJ]

\*/

for (i = 1; i < MAX\_INIT\_ARGS; i++)

argv\_init[i] = NULL;

return 1;

}

\_\_setup("init=", init\_setup);//uboot的命令行参数将传递给execute\_command 值

if (execute\_command) // 有定义execute\_command就执行

{

run\_init\_process(execute\_command);

printk(KERN\_WARNING "Failed to execute %s. Attempting "

"defaults...\n", execute\_command);

}

run\_init\_process("/sbin/init");//没有定义execute\_command就执行这些应用程序

run\_init\_process("/etc/init");

run\_init\_process("/bin/init");

run\_init\_process("/bin/sh");

ls –l /bin/ls /bin/ls- >busybox 最后都是链接到busybox

busybox ls 与 ls 的效果一样

分析busybox:

内核启动的第一个进程是init进程（sbin/init）init启动用户程序。

Init进程要启动应用程序。

Init:

1要读取配置文件 /dev/consol ;/dev/null

2解析配置文件 /etc/inittab

3执行用户程序

4.库

Init本身 就是busybox

以上就是最小的根文件系统

所以：

1./dev/consol ;/dev/null

2.一般来源busybox //启动应用程序run\_init\_process("/sbin/init");

3./etc/inittab //程序读取配置

4.配置文件指定的应用程序

5.c库函数提供一些函数

在busybox中：

首先是init\_main

parse\_inittab();

file = fopen(INITTAB, "r");//打开配置文件

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

实际的执行：new\_init\_action(int action, -/bin/sh, “dev/tty2”);//分别对应配置中的三个有用的参数

1．创建一个init\_action结构，填充

2. 把init\_action放入init\_action\_list链表

run\_actions(SYSINIT);//最先执行

waitfor(a, 0);//执行应用程序等他执行完毕

run(a)//执行应用程序等他执行完毕

waitpid(runpid, &status, 0);//等待他结束

delete\_init\_action(a);// 在init\_action\_list删除

run\_actions(WAIT);//然后

waitfor(a, 0);//执行应用程序等他执行完毕

run(a)//执行应用程序等他执行完毕

waitpid(runpid, &status, 0);//等待他结束

delete\_init\_action(a);// 在init\_action\_list删除

run\_actions(ONCE);//再然后

run(a);//创建progress进程

"\nPlease press Enter to activate this console. ";

delete\_init\_action(a);

while(1)

{

run\_actions(RESPAWN);

if (a->pid == 0) {

a->pid = run(a);

}

/\* run the askfirst stuff \*/

run\_actions(ASKFIRST);

if (a->pid == 0) {

a->pid = run(a);

}

wpid = wait(NULL);//等待子进程退出

while (wpid > 0)

{

a->pid = 0;//退出后就设置pid =0

}

}

从默认的new\_init\_action反推出默认的配置文件

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

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

// ctrlaltdel reboot

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

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

// shutdown umount -a -r

/\* Swapoff on halt/reboot \*/ /\*这一项对于我们嵌入式基本没什莫用\*/

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

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

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

// 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);

// tty2::ASKFIRST: -/bin/sh

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

// tty3::ASKFIRST: -/bin/sh

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

// tty4::ASKFIRST: -/bin/sh

/\* sysinit \*/

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

// :: SYSINIT: /etc/init.d/rcS

new\_init\_action(ASKFIRST, -/bin/sh, “dev/tty2”)

// ASKFIRST:-/bin/sh

inittab 配置文件的格式可以发现如下的格式

<id>:<runlevels>:<action>:<process>

Id =>/dev/id 用作终端 stdin stdout stderr printf scanf

<runlevels> ：忽略

Action ；何时执行

<action>: Valid actions include: sysinit, respawn, askfirst, wait, once,

restart, ctrlaltdel, and shutdown.

Process 可以执行的程序或者脚本

Busybox：

1执行 make menuconfig

有配置项

或者在makefile 中

CROSS\_COMPILE ?= arm-linux- //加上

1. make

3 . mkdir -p /work/nfs\_root/first\_fs

make CONFIG\_PREFIX=/work/nfs\_root/first\_fs install

//不要直接make install 他会安装到windows下 要新建一个目录

创建根文件系统：

根据以下的文件

1./dev/consol ;/dev/null

2.一般来源busybox //启动应用程序run\_init\_process("/sbin/init");

3./etc/inittab //程序读取配置

4.配置文件指定的应用程序

5.c库函数提供一些函数

1./dev/consol ;/dev/null

1．在上述make install新建的目录下

2. 新建一个目录

3. sudo mknod console c 5 1

4. sudo mknod null c 1 3

3. /etc/inittab //程序读取配置

在此目录下新建一个文件夹etc

新建一个inittab文件输入内容console::askfirst:-/bin/sh

5.c库函数提供一些函数

新建文件夹mkdir /work/nfs\_root/first\_fs/lib

cp \*.so\* /work/nfs\_root/first\_fs/lib/ -d 拷贝所有的动态库到目录下 要加-d 选项否则所指的链接文件就会被拷贝过来 库非常大

这样first\_fs 就是最小的根文件系统 如何烧写到板子上？

1制作映像（我们用的是每页2048 用yaffs2）

进入/work/system 找到yaffs\_source\_util\_larger\_small\_page\_nand.tar 压缩包解压

进入system/Development\_util\_ok/yaffs2/utils$ make

得到mkyaffs2image mkyaffsimage 工具

sudo cp mkyaffs2image /usr/local/bin/

sudo chmod +x /usr/local/bin/mkyaffs2image //加上权限

进入存放first\_fs的目录 然后mkyaffs2image first\_fs first\_fs.yaffs2 得到first\_fs.yaffs2就可以烧到板子上了（通过dnw）

2.假如我们执行ps命令 肯定会找不到

我们在串口的界面 启动我们的system (b选项)

然后新建目录 mkdir proc

Mount –t proc none /proc //应该手动挂载之类的

这样就能查看ps 命令 （proc是内核提供的虚拟文件系统）

我们可以直接将proc做进我们的根文件系统这样我们就不用手动挂载了

1. Vi etc/inittab 再添加下面的一行

加上:: sysinit: /etc/init.d/rcS

创建文件mkdir etc/init.d vi etc/init.d/rcS 添加 mount –t proc none /proc 添加权限 chmod +x /etc/init.d/rcS

2. 也可以用rcS 脚本执行 mount –a 命令 它去读出/etc/fstab，也就是依赖fstab去挂载文件系统

Fastab：格式 device mount\_point type options dump fsck order

Vi /etc/fastab proc /proc defaults 0 0

Cat etc/init.d/rcS 会有mount –a 产生 （我的第二种没有成功）

Udev =>> 自动创建dev的设备节点

Mdev >>udev的简化版本（可以在busybox下查找）

如下：

Here's a typical code snippet from the init script:

[1] mount -t sysfs sysfs /sys

[2] echo /bin/mdev > /proc/sys/kernel/hotplug

[3] mdev -s

Of course, a more "full" setup would entail executing this before the previous

code snippet:

[4] mount -t tmpfs mdev /dev

[5] mkdir /dev/pts

[6] mount -t devpts devpts /dev/pts

在原来的proc /proc proc defaults 0 0 后加上

sysfs /sys sysfs defaults 0 0

tmpfs /dev tmpfs defaults 0 0

在rcS启动脚本加上：

Mount –a

Mkidr /dev/pts

Mount –t devpts devpts /dev/pts

Echo /sbin/mdev > /proc/sys/kernel/hotplug

Mdev –s//将内核原来的驱动列出

1. Jff2s的制作

没找到zlib-1.2.3

我们每次制作完根文件系统都要进行烧写有什莫办法不用烧写？

启动后可以将服务器的NFS的当作根文件系统

我找到 fs\_mini.jffs2 或者 烧进去 （这样是还是yaffs2启动）

OpenJTAG> set bootargs noinitrd root=/dev/mtdblock3 rootfstype=jffs2 init=/linuxrc console=ttySAC0

OpenJTAG> save

OpenJTAG> boot 重新启动

就会从jiffs2启动

设置ip 与我们的ubuntun可以ping通 ifconfig eth0 192.168.0.11 （要在一个网段）

1. 从flash上启动根文件系统，用命令挂接nFS

1挂载nfs 首先服务允许那个目录可以被挂接 NFS服务 /etc/exports

假如我们希望挂接/work/nfs\_root/first\_fs$这个目录

Sudo vi /etc/exports 仿照加上/work/nfs\_root/first\_fs

sudo /etc/init.d/nfs-kernel-server restart

自己挂接自己sudo mount -t nfs 192.168.0.105:/work/nfs\_root/first\_fs /mnt

2然后单板去挂接

mount –t nfs -o nolock 192.168.0.105:/work/nfs\_root/first\_fs /mnt

这样在first\_fs目录下的的温江就变成了共享的

1. 从nfs启动

修改命令行参数

在命令行参数可以找到如下：

bootargs=noinitrd root=/dev/mtdblock3 rootfstype=jffs2 init=/linuxrc console=ttySAC0

如果我们修改bootargs=noinitrd root=/dev/nfss rootfstype=jffs2 init=/linuxrc console=ttySAC0 就可以从网络盘启动

Nfs 参数设置包括 服务器ip 目录等

格式的话去网上找 root=/dev/nfs参数

可以在内核中搜索 grep “nfsroot=” \* -nR 其实在内核文档nfsroot

Set bootargs Noinitrd root=/dev/nfs nfsroot=192.168.0.105: /work/nfs\_root/first\_fs

ip=192.168.0.11:192.168.0.105:192.168.0.1:255.255.255.0::eth0:off init=/linuxrc console=ttySAC0

/work/nfs\_root/first\_fs init=/linuxrc console=ttySAC0

最后得到如下：

在命令行参数输入

set bootargs noinitrd root=/dev/nfs nfsroot=192.168.0.105:/work/nfs\_root/first\_fs ip=192.168.0.11:192.168.0.105:192.168.0.1:255.255.255.0::eth0:off init=/linuxrc console=ttySAC0

save

boot启动

这样我们在服务器上的文件可以直接在版上执行