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

要制作根文件系统：

在挂接根文件系统之前，需要制作根文件系统。根文件系统里面需要一些基本的命令，目录和设备文件，下面来介绍如何使用busybox来制作根文件系统。

      我们都知道，init进程是系统启动后执行的第一个应用程序，根据一般的Linux应用程序配置结构，一个可执行文件通常搭配一个对于的配置文件，例如samba功能对于/etc/samba/smb.conf配置文件，nfs功能对于/dev/exports配置文件，那么init初始化功能肯定也是对于一个配置文件，这个配置文件叫做/etc/inittab.配置文件根据不同的级别，指定在不同的情况下，执行不同的应用程序。

<https://www.cnblogs.com/cherishui/p/4237694.html//>根文件系统制作介绍

首先我们板子上下载了uboot和我们的kernel image

然后用busybox制作:

Busybox：

一 执行 make menuconfig

如有makefile报错可以参考：

<https://www.cnblogs.com/weidongshan/p/8036182.html>

<https://www.cnblogs.com/lialong1st/p/8051563.html>

我们要配置交叉编译项：如果在make menuconfig 没有此选项

修改makefile

CROSS\_COMPILE ?= arm-linux- //加上

而且做好在make menuconfig中选中补全按键

Linux Module Utilities 也要勾选上

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

之后会在目录下生成：

bin linuxrc sbin usr 这几个目录

/\*

busybox编译的结果有bin目录，sbin目录和usr目录，根目录下面还有一个linuxrc链接文件，指向bin/busybox。查看一下所有生成的文件，他们都是链接文件，均指向/bin/busybox。这样的目录构造对于启动一个正常的linux来说还是不够的，下面继续制作。

\*/

添加其他必须的文件

     添加基本的设备节点，仿照当前系统的console设备节点和null设备节点，在busybox的目录下也建立对于的节点。

二 我们要根据/dev/console /dev/null 自己创建这两个

Mkdir dev

sudo mknod console c 5 1//

sudo mknod null c 1 3//

这样前两步就完成了

3. 构造自己的启动配置文件 /etc/inittab

    为了简单起见，这里面只设置启动时开启shell输入。首先创建etc目录，然后执行如下操作。

三. /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

四.c库函数提供一些函数

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

进去/work/tools/gcc-3.4.5-glibc-2.3.6/arm-linux/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）//yaffs1 用于小页的

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

qbin/

要拷贝到/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）

删除已下载的文件系统：

Nand erase root

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 //注意要新建一个文件夹否则下面的命令执行不成功

mount –t proc none /proc //应该手动挂载之类的

这样就能查看ps 命令 （它去proc 目录。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/ /加上这个脚本使得开机后自动执行rcS脚本文件。

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 //就可以开启自动挂载proc虚拟文件系统

2. 如果需要挂载多个文件系统 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

用mdev创建设备文件：

Udev =>> 自动创建dev的设备节点

3. 使能热插拔功能 mdev

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

创建一个sys目录

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   # devpts支持外部网络连接(telnet)的虚拟终端

mount –t devpts devpts /dev/pts

echo /sbin/mdev > /proc/sys/kernel/hotplug # 设置内核热拔插事件回调程序

mdev –s在/dev目录下，生成内核目前支持的所有设备节点

//除了busybox生成的bin sbin usr linuxrc

其他的目录 例如 proc dev etc都需要自己创建

Lesson\_learn: 我再用mount –a制作的文件系统，仿照视频用ps命令：

一直提示connot read/etc/fstab 究其原因我在自己制作的根文件的目录下并没有创建这个目录。

cat /proc/mounts //就可以看到它有很多的文件系统了

假如在jff2s文件系统项恢复到yaff2s文件系统用如下：

set bootargs console=ttySAC0,115200 root=/dev/mtdblock3

**用jff2s启动**

set bootargs noinitrd root=/dev/mtdblock3 rootfstype=jffs2 init=/linuxrc\_console=ttySAC0

Jff2s的制作：因为jff2s是压缩的文件系统，zlib里面会有很多的库

1. 首先找到zlib-1.2.3工具 //我的再/work/system目录下
2. 解压后配置 ./configure –shared –prefix=/usr/ //生成动态库。安装再usr目录下
3. sudo make intall // 会安装再system目录下
4. 找到mtd-utils-05.07.23 //我的在/work/tools/
5. 进入mtd-utils-05.07.23/util
6. make
7. sudo make install
8. 得到一个工具mkfs.jffs2
9. mkfs.jffs2 -n -s 2048 -e 128KiB -d first\_fs -o first\_fs.jffs2//一页是2048 每个块128k –d表示目录//一定要注意KIB的大小写
10. 你会发现还是以yaffs2挂接的// VFS: Mounted root (yaffs filesystem).
11. 所以我们要修改指定根文件系统类型
12. set bootargs noinitrd root=/dev/mtdblock3 rootfstype=jffs2 init=/linuxrc console=ttySAC0//视频中的命令
13. set bootargs console=ttySAC0,115200 root=/dev/mtdblock3 rootfstype=jffs2//答疑 的解答
14. save
15. boot

**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？**

一：可以用网络文件系统，这个文件是放在服务器上的，内核启动后自己去读取这个文件系统

启动后可以将服务器的NFS的当作根文件系统：

**NFS是什么？**

System（网络文件系统），NFS只是一种文件系统，本身并没有传输功能，是基于RPC协议实现的，为C/S[**架构**](https://link.zhihu.com/?target=http%3A//lib.csdn.net/base/architecture)。也就是说，NFS也只是文件系统而已，与jffs2,yaffs,ext4等是一类东西。但是NFS的特殊之处在于，它可以通过网络来挂载，而不用烧写到设备上。这就跟网吧里使用PXE启动来搭建无盘系统一样：每台电脑并没有安装硬盘，而是直接通过PXE启动服务器上的系统，看起来每台电脑都有硬盘，其实都是挂载服务器上的硬盘。

**FS怎么用？** 前边已经介绍过，NFS是C/S架构，也就是说，使用NFS需要搭建服务器和客户端。客户端很简单，自带的uboot已经添加了对NFS客户端的支持，我们只需要管NFS服务器。这个服务器其实跟TFTP、Samba、 Ftp服务器是一类的东西，直接在Ubuntu上使用命令安装即可

这手动和自动挂载的区别在哪里？

1. 仅用flash上的根文件系统启动，手工MOUNT NFS 介绍完毕 强烈推荐这种方式，因为简单。随时想挂载就挂载，Ubuntu系统即使不开机(前提是ubuntu ip没变)也不影响开发板的启动。
2. 而 2、使用NFS作为根文件系统来启动，若Ubuntu未启动或者文件系统有问题，开发板根本无法开机，严重影响体验。

首先在板子的串口ifconfig

不OK的话，ifconfig eth0 up

设置我们板子的ip:如ifconfig eth0 192.168.0.11

ping 我们的服务器

1. 从flash上启动根文件系统，**手工MOUNT 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//可以先试着将自己挂接自己将我们需要的目录挂接到/mnt下

1. 然后单板去挂接

首先新建一个文件夹 mkdir /mnt

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

在服务器下操作echo hello > test.txt 就可以在板子上看见了cat /mnt/test.txt

这种是板子启动后我们手动去做的，假如我们要从启动的时候就做好呢。修改命令行参数

1. 使用NFS作为根文件系统启动

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

具体可以参考uboot\_set\_nfs\_启动这个txt文本

boot启动

这样我们在服务器上的文件可以直接在版上执行

假如我们的自动挂载nfs启动不能正常要检查set uboot参数是否正确