以下假设内核源码路径为：/usr/src/linux

一、initramfs是什么

1. cpio格式的打包文件。

2. 内核启动时将其挂载为rootfs，并执行其中的init。如果其中没有init，则正常地挂载根文件系统并执行/sbin/init。

3. 编译2.6版本的linux内核时，编译系统总会创建initramfs，然后把它与编译好的内核连接在一起。

3. 内核源代码树中的usr目录就是专门用于构建内核中的initramfs的。

二、构建第一个initramfs：hello world

mkdir image

image

emacs hello.c

#include <stdio.h>

#include <stdlib.h>

int main(int argc,char argv[])

{

printf("hello world, from initramfs.\n");

sleep(9999999);

return 0;

}

gcc -o init -static -s hello.c # -s选项取消调试信息

mkdir dev

mknod -m 600 dev/console c 5 1

编译内核：

make menuconfig

General setup

--> Initramfs source file(s)

Executable file formats / Emulations

--> Kernel support for ELF binaries

make

make modules\_install

三、试验环境搭建

采用qemu将上面编译出的内核在虚拟机中运行。

qemu -kernel arch/x86/boot/bzImage -vnc :1

四、什么是rootfs和ramfs

1. rootfs是内核启动的初始始根文件系统。

2. initramfs的文件会复制到rootfs。

3. 内核正常运行后都会安装另外的文件系统，然后忽略rootfs。

4. rootfs是ramfs文件系统的一个特殊实例。

5. ramfs是一种基于内存的文件系统。

1. 没有容量大小的限制。

2. 利用了内核的磁盘高速缓存机制。

3. 没有对应文件系统设备，所以它的数据永远都不会回写，因此系统永远不会释放ramfs所占用的内存。

6. 与ramdisk比较

1. ramdisk是内存中的块设备。

2. ramdisk的容量是固定的。

3. 需要内核的文件系统驱动程序。

4. 与高速缓存之间不必要的拷贝。

7. ramfs 的一个缺点是它可能不停的动态增长直到耗尽系统的全部内存，所以从ramfs派生出了 tmpfs文件系统，增加了容量大小的限制，而且允许把数据写入交换分区。

七、什么是busybox

八、busybox的配置、编译和安装

1. 下载busybox-1.19.3.tar.bz2：http://busybox.net/downloads/busybox-1.19.3.tar.bz2

tar jxvf busybox-1.19.3.tar.bz2

cd busybox-1.19.3

2. 配置busybox

make defconfig # 选择所有applet

make allyesconfig # 最大配置，与defconfig相同

make allnoconfig # 最小配置

make menuconfig # 定制配置

3. 编译busybox

make

4. 安装busybox

make install CONFIG\_PREFIX=~/image

cd ~/image

5. 复制必要的动态链接库，因为busybox在编译时默认地做动态链接。

mkdir lib

cp /lib/ld-linux.so.2 lib

cp /lib/libc.so.6 lib

cp /lib/libcrypt.so.1 lib

cp /lib/libm.so.6 lib

九、在image下创建必要的目录和设备文件

mkdir proc sys etc mnt

mknod -m 600 dev/null c 1 3

十、试验一下

mount -vt proc proc proc

mount -vt sysfs sysfs sys

mount -v -o bind /dev dev

chroot . /bin/sh

十一、自动生成/dev下的设备文件

1. 静态生成设备文件的方法不够灵活。

2. busybox中有一个mdev命令，用来动态生成设备文件，填充到/dev目录。在系统启动时，用mdev -s

命令可以根据内核的sysfs文件系统在/dev目录中自动生成相应的设备文件。

3. 命令执行前，需要先挂载内核的proc和sysfs虚拟文件系统。

十二、初始身手

1. 在image目录下写一个最简单的init脚本。

#!/bin/sh

mount -t proc proc /proc

mount -t sysfs sysfs /sys

mdev -s

/bin/sh

2. 为init脚本设置可执行权限，否则内核不会去执行它。

chmod +x init

3. 为了避免出错信息，我们创建一个空/etc/mdev.conf文件。

touch etc/mdev.conf

4. 编译内核

cd /usr/src/linux

make

十三、can't access tty

1. 原因是我们的SHELL是直接运行在内核的console上的，而console是不能提供控制终端（terminal）功能的，所以必须把 SHELL运行在tty设备上，才能消除这个错误。

2. 这个简单系统的reboot、halt等命令是不起作用的。

3. 解决问题的办法是使用正规init机制。

十四、busybox的缺省init模式

不能解决问题。

十五、busybox的inittab文件格式说明

1. 要解决问题需要编写inittab文件。

2. inittab文件各字段的含义。

/etc/inittab 文件中每个条目用来定义一个子进程，并确定它的启动方法，格式如下 ：

:::

例如：

ttySAC0:askfirst:-/bin/sh

（1）:表示这个进程要使用的控制台（即标准输入、标准输出、标准错误设备）。如果省略，则使用与init进程一样的控制台。

（2）:对于Busybox init程序，这个字段滑意义，可以省略。

（3）:表示init程序如何控制这个子进程，

（4）: 要执行的程序，它可以是可执行程序，也可以是脚本

如果:字段有"-"字符，表示这个程序被称为“交互的”。在/etc/inittab/文件的控制下，init进程的行为总结如下：

（1）在系统启动前期，init进程首先启动为sysinit wait once的3类子进程。

（2）在系统正常运行期间，init程序首先启动,为respawn askfirst的两类子进程，并监视它们，发现某个子进程退出时重新启动它。

（3） 在系统退出时，执行 为shutdown restart ctrlaltdel的3类子进程之一或全部。

如果根文件系统中没有/etc/initab文件，Busybox init程序将使用如下默认的inittab条目。

/etc/inittab文件中字段的意义

|  |  |  |
| --- | --- | --- |
| Action名称 | 执行条件 | 说明 |
| Sysinit | 系统启动后最先执行 | 只执行一次，init进程等待它结束才继续执行其它动作 |
| Wait | 系统执行完sysinit进程后 | 只执行一次，init进程等待它结束才继续执行其它动作 |
| Once | 系统执行完wait进程后 | 只执行一次,init进程不等待它结束 |
| Respawn | 启动完once进程后 | Init进程监测发现子进程退出时，重新启动它 |
| Askfirst | 启动完respawn进程后 | 与respawn类似，不过init进程先输出“Please press Enter to actvie this console”，等用户输入回车键之后才启动子进程 |
| Shutdown | 当系统关机时 | 即重启关闭系统命令时 |
| Restart | Busybox中配置了CONFIG\_FEATURE\_USE\_INITTAB，并且init进程接收到SIGHUP信号时 | 先重新读取，解析/etc/initab文件，再执行restart程序 |
| Ctrlatldel | 按下Ctr+Alt+del组合键时 |  |

十六、写mini linux的inittab

cd ~/image

emacs init # 将其中的最后一行删除

mv init etc/init.d/rcS

mv linuxrc init # linuxrc是/bin/busybox的符号链接

emacs etc/inittab

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

tty1::askfirst:/bin/sh

tty2::askfirst:/bin/sh

tty3::askfirst:/bin/sh

tty4::askfirst:/bin/sh

tty5::askfirst:/bin/sh

tty6::askfirst:/bin/sh

::restart:/sbin/init

::ctrlaltdel:/sbin/reboot

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

touch etc/fstab

cd /usr/src/linux

make

qemu -kernel arch/x86/boot/bzImage -vnc :1

十七、配置内核支持initrd

1. 用cpio命令生成单独的initramfs，与内核脱钩并在内核运行时以initrd的形式加载到内核。

2. 用内核usr目录中编译出的initrd做测试：

cp usr/initramfs\_data.cpio ~

make menuconfig

首先配置内核使用单独的initrd:

Device Drivers

--> Block devices

--> RAM block device support

注：此时，即使将initramfs编译到内核的同时，也会在usr目录中生成相应的initrd。

清除原先initramfs的内容：

General Setup

--> initramfs source file(s)

make

qemu -kernel arch/x86/boot/bzImage -initrd ~/initramfs\_data.cpio -vnc :1

十八、用cpio命令生成initramfs

将image中的内容打包成initramfs：

cd ~/image

find . | cpio -o -H newc | gzip > ~/image.cpio.gz

cd /usr/src/linux

qemu -kernel arch/x86/boot/bzImage -initrd ~/image.cpio.gz -vnc :1

十九、cpio命令的其他用法

cpio -i -F ~/image.cpio.gz --no-absolute-filename # 解开cpio文件

zcat ~/image.cpio.gz | cpio -t # 查看cpio文件的内容

二十、switch\_root 命令

1. 通常initramfs都是为安装最终的根文件系统做准备工作，它的最后一步需要安装最终的根文件系统，然后切换到新根文件系统上去。

2. initrd 使用pivot\_root命令切换到新的根文件系统，然后卸载ramdisk。

3. initramfs是rootfs，而rootfs既不能 pivot\_root，也不能umount。

4. busybox的解决方案switch\_root命令。

5. switch\_root命令的格式是：

switch\_root [-c /dev/console] NEW\_ROOT NEW\_INIT [ARGUMENTS\_TO\_INIT]

其中，

（1）NEW\_ROOT是实际的根文件系统的挂载目录；

（2）NEW\_INIT是实际根文件系统的init程序的路径，一般是/sbin/init；

（3）-c /dev/console是可选参数，用于重定向实际的根文件系统的设备文件，一般情况我们不会使用；

（4）ARGUMENTS\_TO\_INIT则是传递给实际的根文件系统的init程序的参数，也是可选的。

例如：exec switch\_root /mnt /sbin/init # /mnt上挂载了一个文件系统

注意：switch\_root命令必须由PID=1的进程调用，也就是必须由initramfs的init程序直接调用，不能由init派生的其他进程调用，否则会出错，提示：

switch\_root: not rootfs。可以再init脚本中加入：exec /bin/sh，使shell的PID=1，从而可以执行switch\_to命令。

二十一、实践：用initramfs安装CLFS根文件系统

（略）

二十二、内核模块支持

1. 首先，内核配置要支持模块，并支持内核模块的自动加载功能：

Load module support

--> Enable loadable module support

--> Automatic kernel loading

2. 然后把需要的硬件驱动程序配置模块形式（例如对SCSI硬盘的支持）。

3. 编译内核，并把编译好的内核模块安装到image的目录下：

make

make INSTALL\_MOD\_PATH=~/image modules\_install

二十三、试验：用initramfs中的内核模块安装硬盘文件系统

1. 测试SCSI硬盘的支持。

2. 内核并没有自动加载硬盘控制器的驱动程序，所以 /dev目录下也没有sda等硬盘设备文件。

3. 查看/lib/modules /2.6.17.13/modules.dep，弄清楚了4个模块的依赖关系。

4. busybox的modprobe命令执行不正常，用insmod命令依次加载。

5. mdev -s 命令生成硬盘的设备文件。

二十四、mdev的hotplug模式

1. 上面加载完驱动模块后调用了mdev -s 命令来生成硬盘的设备文件。

2. 可以使用mdev的hotplug模式：echo /sbin/mdev > /proc/sys/kernel/hotplug，设置系统的hotplug程序为mdev。

3. 后续使用insmod命令加载模块时，系统自动调用mdev生成相应的设备文件。

二十五、udev的coldplug模式

1. 内核在启动时已经检测到了系统的硬件设备，并把硬件设备信息通过sysfs内核虚拟文件系统导出。

2. udev扫描sysfs文件系统，根据硬件设备信息生成热插拔（hotplug）事件，udev再读取这些事件，生成对应的硬件设备文件。

3. 在initramfs的init脚本中添加：

二十六、试验：用udev自动加载设备驱动模块

1. 从 /sbin 目录下拷贝udevd、udevtrigger、udevsettle程序到image目录下的sbin目录下，并用ldd命令找到它们所需要的动态库文件，拷贝到image目录下的lib目录下。

2. 修改init脚本，增加coldplug功能：

#!/bin/sh

mount -t proc proc /proc

mount -t sysfs sysfs /sys

mdev -s

#using udev autoload hard disk driver module

mkdir -p /dev/.udev/db

udevd --daemon

mkdir -p /dev/.udev/queue

udevtrigger

udevsettle

mount /dev/sda8 /mnt

killall udevd

exec switch\_root /mnt /sbin/init

注意：在切换到真正根文件系统前，要把udevd进程杀掉，否则会和真正根文件系统中的udev脚本的执行相冲突。这就是上面killall udevd 语句的作用。

3. 编写udev规则文件。

4. 拷贝modprobe命令及其依赖的动态链接库。因为busybox的modprobe有问题。