## 1、什么是 initrd 以及为什么要用 initrd

在早期的 Linux 系统中，一般就只有软盘或者硬盘被用来作为 Linux 的根文件系统，因此很容易把这些设备的驱动程序集成到内核中。但是现在根文件系统可能保存在各种存储设备上，包括 SCSI, SATA, U 盘等等。因此把这些设备驱动程序全部编译到内核中显得不太方便。我们看到利用 udevd 可以实现实现内核模块的自动加载，因此我们希望根文件系统的设备驱动程序也能够实现自动加载。但是这里有一个矛盾，udevd 是一个可执行文件，在根文件系统被挂载前，是不可能执行 udevd 的，但是如果 udevd 没有启动，那就无法自动加载根根据系统设备的驱动程序，同时也无法在/dev 目录下建立相应的设备节点。为了解决这个矛盾，于是出现了 initrd(boot loader initialized RAM disk)。initrd 是一个被压缩过的小型根目录，这个目录中包含了启动阶段中必须的驱动模块，可执行文件和启动脚本。包括上面提到的 udevd，当系统启动的时候，booload 会把 initrd 文件读到内存中，然后把 initrd 的起始地址告诉内核。内核在运行过程中会解压 initrd，然后把 initrd 挂载为根目录，然后执行根目录中的/initrc 脚本，您可以在这个脚本中运行 initrd 中的 udevd，让它来自动加载设备驱动程序以及在/dev 目录下建立必要的设备节点。在 udevd 自动加载磁盘驱动程序之后，就可以 mount 真正的根目录，并切换到这个根目录中。

启动时用到 initrd 来 mount 根文件系统。其实 ramdisk 只是在 ram 上实现的块设备，initrd 可以说是启动过程中用到的一种机制

## 2、Ramdisk 与 initrd 区别

ramdisk 是一种基于内存的虚拟文件系统，就好像你又有一个硬盘，你可以对它上面的文件添加修改删除等等操作。但是一掉电，就什么也没有了。无法保存。
initrd 是 boot loader initialized RAM disk 顾名思义，是在系统初始化引导时候用的 ramdisk。也就是由启动加载器所初始化的 RamDisk 设备，它的作用是完善内核的模块机制，让内核的初始化流程更具弹性；内核以及 initrd，都由 bootloader 在机子启动后被加载至内存的指定位置，主要功能为按需加载模块以及按需改变根文件系统。

## 3、Linux 启动一定要用 initrd 么?

如果把需要的功能全都编译到内核中(非模块方式)，只需要一个内核文件即可。initrd 能够减小启动内核的体积并增加灵活性，如果你的内核以模块方式支持某种文件系统(例如 ext3, UFS)，而启动阶段的驱动模块放在这些文件系统上,内核是无法读取文件系统的，从而只能通过 initrd 的虚拟文件系统来装载这些模块。这里有些人会问：既然内核此时不能读取文件系统，那内核的文件是怎么装入内存中的呢?答案很简单，Grub 是 file-system sensitive 的，能够识别常见的文件系统。

## 4、制作 initrd

**1．Linux2.4 内核对 Initrd 的处理流程**

为了使读者清晰的了解 Linux2.6 内核 initrd 机制的变化，在重点介绍 Linux2.6 内核 initrd 之前，先对 linux2.4 内核的 initrd 进行一个简单的介绍。Linux2.4 内核的 initrd 的格式是文件系统镜像文件，本文将其称为 image-initrd，以区别后面介绍 的 linux2.6 内核的 cpio 格式的 initrd。 linux2.4 内核对 initrd 的处理流程如下：

1. boot loader 把内核以及/dev/initrd 的内容加载到内存，/dev/initrd 是由 boot loader 初始化的设备，存储着 initrd。

2. 在内核初始化过程中，内核把 /dev/initrd 设备的内容解压缩并拷贝到 /dev/ram0 设备上。

3. 内核以可读写的方式把 /dev/ram0 设备挂载为原始的根文件系统。

4. 如果 /dev/ram0 被指定为真正的根文件系统，那么内核跳至最后一步正常启动。

5. 执行 initrd 上的 /linuxrc 文件，linuxrc 通常是一个脚本文件，负责加载内核访问根文件系统必须的驱动， 以及加载根文件系统。

6. /linuxrc 执行完毕，真正的根文件系统被挂载。

7. 如果真正的根文件系统存在 /initrd 目录，那么 /dev/ram0 将从 / 移动到 /initrd。否则如果 /initrd 目录不存在， /dev/ram0 将被卸载。

8. 在真正的根文件系统上进行正常启动过程 ，执行 /sbin/init。 linux2.4 内核的 initrd 的执行是作为内核启动的一个中间阶段，也就是说 initrd 的 /linuxrc 执行以后，内核会继续执行初始化代码，我们后面会看到这是 linux2.4 内核同 2.6 内核的 initrd 处理流程的一个显著区别。

**2．Linux2.6 内核对 Initrd 的处理流程**

linux2.6 内核支持两种格式的 initrd，一种是前面第 3 部分介绍的 linux2.4 内核那种传统格式的文件系统镜像－image-initrd，它的制作方法同 Linux2.4 内核的 initrd 一样，其核心文件就是 /linuxrc。另外一种格式的 initrd 是 cpio 格式的，这种格式的 initrd 从 linux2.5 起开始引入，使用 cpio 工具生成，其核心文件不再是 /linuxrc，而是 /init，本文将这种 initrd 称为 cpio-initrd。尽管 linux2.6 内核对 cpio-initrd 和 image-initrd 这两种格式的 initrd 均支持，但对其处理流程有着显著的区别，下面分别介绍 linux2.6 内核对这两种 initrd 的处理流程。

cpio-initrd 的处理流程

1． boot loader 把内核以及 initrd 文件加载到内存的特定位置。

2． 内核判断 initrd 的文件格式，如果是 cpio 格式。

3． 将 initrd 的内容释放到 rootfs 中。

4． 执行 initrd 中的/init 文件，执行到这一点，内核的工作全部结束，完全交给/init 文件处理。

image-initrd 的处理流程

1． boot loader 把内核以及 initrd 文件加载到内存的特定位置。

2. 内核判断 initrd 的文件格式，如果不是 cpio 格式，将其作为 image-initrd 处理。

3. 内核将 initrd 的内容保存在 rootfs 下的/initrd.image 文件中。

4. 内核将/initrd.image 的内容读入/dev/ram0 设备中，也就是读入了一个内存盘中。

5. 接着内核以可读写的方式把/dev/ram0 设备挂载为原始的根文件系统。

6. .如果/dev/ram0 被指定为真正的根文件系统，那么内核跳至最后一步正常启动。

7. 执行 initrd 上的/linuxrc 文件，linuxrc 通常是一个脚本文件，负责加载内核访问根文件系统必须的驱动，以及加载根文件系统。

8. /linuxrc 执行完毕，常规根文件系统被挂载

9. 如果常规根文件系统存在/initrd 目录，那么/dev/ram0 将从/移动到/initrd。否则如果/initrd 目录不存在，/dev/ram0 将被卸载。

10. 在常规根文件系统上进行正常启动过程 ，执行/sbin/init。

通过上面的流程介绍可知，Linux2.6 内核对 image-initrd 的处理流程同 linux2.4 内核相比并没有显著的变化， cpio-initrd 的处理流程相比于 image-initrd 的处理流程却有很大的区别，流程非常简单，在后面的源代码分析中，读者更能体会到处理的 简捷。

4．cpio-initrd 同 image-initrd 的区别与优势

没有找到正式的关于 cpio-initrd 同 image-initrd 对比的文献，根据笔者的使用体验以及内核代码的分析，总结出如下三方面的区别，这些区别也正是 cpio-initrd 的优势所在：

cpio-initrd 的制作方法更加简单

cpio-initrd 的制作非常简单，通过两个命令就可以完成整个制作过程

```
#假设当前目录位于准备好的 initrd 文件系统的根目录下
bash# find . | cpio -c -o > ../initrd.img
bash# gzip ../initrd.img
```

**具体过程：在 busybox 中添加配置文件并生成 initrd 镜像**

这时，我们处在/root/busybox-1.16.0/_install 目录下。

# mkdir proc sys etc dev (创建四个空目录，linux 内核需要)

# cd dev

# mknod console c 5 1 (创建一个控制台字符设备文件)

# mknod null c 1 3 (创建一个 0 设备文件)

# cd ..

# cd etc

# vim fstab (输入如下图内容)

```
# device    mount-point    type      options    dump    fsck
proc        /proc          proc      defaults   0       0
sysfs       /sys           sysfs     defaults   0       0
```

# mkdir init.d

# vim init.d/rcS (输入如下内容)

```
#!/bin/sh
mount -a
```

# chmod +x init.d/rcS (给 rcS 文件加上可执行权限)

# vim inittab (输入如下内容)

```
# /etc/inittab
::sysinit:/etc/init.d/rcS
console::respawn:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

# cd ..

# pwd (打印当前目录)

/root/busybox-1.16.0/_install

此时表明我们处在 busybox 安装文件的根目录下

# rm linuxrc (删除 linuxrc 链接文件)

然后新创建一个指向 busybox 文件的链接文件，如下图所示：

```
root:~/busybox-1.16.0/_install # ln -sv bin/busybox init
`init' -> `bin/busybox'
root:~/busybox-1.16.0/_install #
```

我们输入如下图所示命令来创建 initrd 镜像.

```
find . | cpio --quiet -H newc -o | gzip -9 -n > ../initrd.gz
```

# cd ..

# cp initrd.gz /boot

而传统 initrd 的制作过程比较繁琐，需要如下六个步骤

```
#假设当前目录位于准备好的 initrd 文件系统的根目录下
bash# dd if=/dev/zero of=../initrd.img bs=512k count=5
bash# mkfs.ext2 -F -m0 ../initrd.img
bash# mount -t ext2 -o loop ../initrd.img   /mnt
bash# cp -r   * /mnt
bash# umount /mnt
bash# gzip -9 ../initrd.img
```