initrd

Linux初始RAM磁盘（initrd）是在系统引导过程中挂载的一个临时根文件系统，用来支持两阶段的引导过程。initrd文件中包含了各种[可执行程序](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=171768&ss_c=ssc.citiao.link" \t "_blank)和驱动程序，它们可以用来挂载实际的根文件系统，然后再将这个 initrd RAM磁盘卸载，并[释放内存](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=174547028&ss_c=ssc.citiao.link" \t "_blank)。在很多[嵌入式Linux](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7795970&ss_c=ssc.citiao.link)系统中，initrd 就是最终的根文件系统。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **外文名称** | initrd | | **含义** | 一个初始根文件系统 | | |  |  | | --- | --- | | **意义** | 支持两阶段的引导过程 | |

**目  
录**

* 1[英文含义](https://baike.sogou.com/v7941205.htm?fromTitle=initrd#para1)
* 2[所用工具](https://baike.sogou.com/v7941205.htm?fromTitle=initrd#para2)

1英文含义

[编辑](https://baike.sogou.com/Create.e?sp=2&sp=l7941205&sp=1)

什么是初始 RAM 磁盘

初始RAM磁盘（initrd）是在实际根文件系统可用之前挂载到系统中的一个初始根文件系统。initrd与内核绑定在一起，并作为内核引导过程的一部分进行加载。内核然后会将这个 initrd文件作为其两阶段引导过程的一部分来加载模块，这样才能稍后使用真正的文件系统，并挂载实际的根文件系统。

initrd 中包含了实现这个目标所需要的目录和[可执行程序](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=171768&ss_c=ssc.citiao.link)的**最小集合**，例如将内核模块加载到内核中所使用的 insmod 工具。

在桌面或服务器Linux 系统中，initrd 是一个**临时**的文件系统。其生存周期很短，只会用作到真实文件系统的一个桥梁。在没有存储设备的[嵌入式系统](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=53418&ss_c=ssc.citiao.link)中，initrd 是**永久**的根文件系统。本文将对这两种情况进行探索。

initrd 剖析

initrd 映像中包含了支持 [Linux系统](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=807585&ss_c=ssc.citiao.link)两阶段引导[过程所](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7708689&ss_c=ssc.citiao.link)需要的必要可执行程序和系统文件。

根据我们运行的 Linux 的版本不同，创建初始 RAM磁盘的方法也可能会有所不同。在 Fedora Core 3 之前，initrd 是使用loop设备 来构建的。loop设备是一个[设备驱动程序](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7921354&ss_c=ssc.citiao.link" \t "_blank)，利用它可以将文件作为一个块设备挂载到系统中，然后就可以查看这个文件系统中的内容了。在您的内核中可能并没有loop设备，不过这可以通过内核配置工具（make menuconfig）选择 Device Drivers > Block Devices> Loopback Device Support 来启用。我们可以按照下面的方法来查看loop设备的内容（initrd文件的名字可能会稍有不同）：

清单 1. 查看 initrd 的内容（适用于 FC3 之前的版本）

mkdir temp ; cd temp

cp /boot/initrd.img.gz .

gunzip initrd.img.gz

mount -t ext -oloopinitrd.img /mnt/initrd

ls -la /mnt/initrd

这样我们就可以查看 /mnt/initrd 子目录中的内容了，这就代表了 initrd文件的内容。注意，即使您的 initrd[映像文件](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=591185&ss_c=ssc.citiao.link" \t "_blank)不是以 .gz 结尾，它也可能是一个压缩文件，您可以给这个文件添加上 .gz后缀，然后再使用 gunzip 对其进行解压。

从 Fedora Core 3 开始，默认的 initrd 映像变成了一个经过压缩的cpio 归档文件。我们不用再使用loop设备来将 initrd 作为压缩映像进行挂载，而是可以将其作为 cpio 归档文件来使用。要查看cpio 归档文件的内容，可以使用下面的命令：

清单 2. 查看 initrd 的内容（适用于 FC3 及其以后的版本）

mkdir temp ; cd temp

cp /boot/initrd-2.6.14.2.img initrd-2.6.14.2.img.gz

gunzip initrd-2.6.14.2.img.gz

cpio -i -d < initrd-2.6.14.2.img

结果会生成一个很小的根文件系统，如清单 3 所示。在 ./bin 目录中有一组很少但却非常必要的应用程序，包括 nash（即 not a shell，是一个脚本[解释器](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10821718&ss_c=ssc.citiao.link" \t "_blank)）、insmod（用来加载内核模块）和 lvm（[逻辑卷](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=73850678&ss_c=ssc.citiao.link)管理工具）。

清单 3. 默认的 Linux initrd目录结构

ls -la

drwxr-xr-x 10 root root 4096 May 7 02:48 .

drwxr-x--- 15 root root 4096 May 7 00:54 ..

drwxr-xr-x 2 root root 4096 May 7 02:48 bin

drwxr-xr-x 2 root root 4096 May 7 02:48 dev

drwxr-xr-x 4 root root 4096 May 7 02:48 etc

-rwxr-xr-x 1 root root 812 May 7 02:48 init

-rw-r--r-- 1 root root 1723392 May 7 02:45 initrd-2.6.14.2.img

drwxr-xr-x 2 root root 4096 May 7 02:48 lib

drwxr-xr-x 2 root root 4096 May 7 02:48loopfs

drwxr-xr-x 2 root root 4096 May 7 02:48 proc

lrwxrwxrwx 1 root root 3 May 7 02:48 sbin -> bin

drwxr-xr-x 2 root root 4096 May 7 02:48 sys

drwxr-xr-x 2 root root 4096 May 7 02:48 sysroot

#

清单 3 中比较有趣的是 init 文件就在根目录中。与传统的 Linux 引导过程类似，这个文件也是在将 initrd 映像解压到 RAM磁盘中时被调用的。在本文稍后我们将来探索这个问题。

2所用工具

[编辑](https://baike.sogou.com/Create.e?sp=2&sp=l7941205&sp=2)

cpio 命令

使用 cpio 命令，我们可以对 cpio 文件进行操作。cpio是一种文件格式，它简单地使用[文件头](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7895355&ss_c=ssc.citiao.link" \t "_blank)将一组文件串接在一起。cpio 文件格式可以使用 ASCII 和[二进制文件](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8417547&ss_c=ssc.citiao.link" \t "_blank)。为了保证可移植性，我们可以使用ASCII 格式。为了减小文件大小，我们可以使用二进制的版本。

下面让我们回到最开始，来看一下 initrd 映像最初是如何构建的。对于传统的Linux系统来说，initrd 映像是在 Linux 构建过程中创建的。有很多工具，例如mkinitrd，都可以用来使用必要的库和模块自动构建 initrd，从而用作与真实的根文件系统之间的桥梁。mkinitrd工具实际上就是一个 [shell脚本](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8961472&ss_c=ssc.citiao.link)，因此我们可以看到它究竟是如何来实现这个结果的。还有一个 YAIRD（即 Yet AnotherMkinitrd）工具，可以对 initrd 构建过程的各个方面进行定制。

手工构建定制的初始 RAM 磁盘

由于在很多基于 Linux 的嵌入式系统上没有硬盘，因此 initrd也会作为这种系统上的永久根文件系统使用。清单 4 显示了如何创建一个 initrd 映像文件。我使用了一个标准的 [Linux桌面](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=139349301&ss_c=ssc.citiao.link)，这样您即使没有嵌入式平台，也可以按照下面的步骤来执行了。除了[交叉编译](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7869994&ss_c=ssc.citiao.link)，其他概念（也适用于 initrd 的构建）对于嵌入式平台都是相同的。

清单 4. 创建定制 initrd 的工具（mkird）

#!/bin/bash

# Housekeeping...

rm -f /tmp/ramdisk.img

rm -f /tmp/ramdisk.img.gz

# Ramdisk Constants

RDSIZE=4000

BLKSIZE=1024

# Create an emptyramdisk image

dd if=/dev/zero of=/tmp/ramdisk.img bs=$BLKSIZE count=$RDSIZE

# Make it an [ext2](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=71706238&ss_c=ssc.citiao.link) mountable file system

/sbin/[mke2fs](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=70227927&ss_c=ssc.citiao.link) -F -m 0 -b $BLKSIZE /tmp/ramdisk.img $RDSIZE

# Mount it so that we can populate

mount /tmp/ramdisk.img /mnt/initrd -t ext2 -oloop=/dev/loop0

# Populate the [filesystem](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=76677035&ss_c=ssc.citiao.link" \t "_blank) (subdirectories)

mkdir /mnt/initrd/bin

mkdir /mnt/initrd/sys

mkdir /mnt/initrd/dev

mkdir /mnt/initrd/proc

# Grabbusyboxand create the symbolic links

pushd /mnt/initrd/bin

cp /usr/local/src/busybox-1.1.1/busybox.

ln -sbusyboxash

ln -sbusyboxmount

ln -sbusyboxecho

ln -sbusyboxls

ln -sbusyboxcat

ln -sbusyboxps

ln -sbusyboxdmesg

ln -sbusyboxsysctl

popd

# Grab the [necessary](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=82174979&ss_c=ssc.citiao.link) dev files

cp -a /dev/console /mnt/initrd/dev

cp -a /dev/ramdisk /mnt/initrd/dev

cp -a /dev/ram0 /mnt/initrd/dev

cp -a /dev/null /mnt/initrd/dev

cp -a /dev/tty1 /mnt/initrd/dev

cp -a /dev/tty2 /mnt/initrd/dev

# Equate sbin with bin

pushd /mnt/initrd

ln -s bin sbin

popd

# Create the init file

cat >> /mnt/initrd/linuxrc << EOF

#!/bin/ash

echo

echo "Simple initrd is active"

echo

mount -t proc /proc /proc

mount -t sysfs none /sys

/bin/ash --login

EOF

chmod +x /mnt/initrd/linuxrc

# Finish up...

umount /mnt/initrd

gzip -9 /tmp/ramdisk.img

cp /tmp/ramdisk.img.gz /boot/ramdisk.img.gz

initrd Linux 发行版

Minimax 是一个开放源码项目，其设计目标是成为一个全部封装在 initrd 中的Linux 发行版。它的大小是 32MB，为了尽量小，它使用了 BusyBox 和 uClibc。除了非常小之外，它还使用了 2.6 版本的[Linux内核](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=374709&ss_c=ssc.citiao.link" \t "_blank)，并提供了很多有用的工具。

为了创建 initrd，我们最开始创建了一个空文件，这使用了/dev/zero（一个由零组成的码流）作为输入，并将其写入到ramdisk.img 文件中。所生成的文件大小是 4MB（4000 个 1K大小的块）。然后使用 mke2fs 命令在这个空文件上创建了一个 ext2（即 secondextended）文件系统。现在这个文件变成了一个 ext2 格式的文件系统，我们使用loop设备将这个文件挂载到 /mnt/initrd上了。在这个[挂载点](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=53281240&ss_c=ssc.citiao.link" \t "_blank)上，我们现在就有了一个目录，它以 ext2文件系统的形式呈现出来，我们可以对自己的 initrd文件进行拼装了。接下来的脚本提供了这种功能。

下一个步骤是创建构成根文件系统所需要的子目录：/bin、/sys、/dev 和 /proc。这里只列出了所需要的目录（例如没有库），但是其中包含了很多功能。

ext2文件系统的替代品

尽管 ext2 是一种通用的 [Linux文件系统](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7867168&ss_c=ssc.citiao.link)格式，但是还有一些替代品可以减小initrd映像文件以及所挂载上来的文件系统的大小。这种文件系统的例子有 romfs（ROM文件系统）、cramfs（压缩 ROM文件系统）和 squashfs（高度压缩[只读文件](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=557431&ss_c=ssc.citiao.link" \t "_blank)系统）。如果我们需要暂时将数据写入文件系统中，ext2可以很好地实现这种功能。最后，e2compr 是 ext2文件系统驱动程序的一个扩展，可以支持在线压缩。

为了可以使用根文件系统，我们使用了BusyBox。这个工具是一个单一映像，其中包含了很多在 Linux系统上通常可以找到的工具（例如 ash、awk、sed、insmod等）。BusyBox的优点是它将很多工具打包成一个文件，同时还可以共享它们的通用元素，这样可以极大地减少映像文件的大小。这对于嵌入式系统来说非常理想。将BusyBox 映像从自己的源目录中拷贝到自己根目录下的 /bin 目录中。然后创建了很多[符号链接](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10823896&ss_c=ssc.citiao.link" \t "_blank)，它们都指向 BusyBox工具。BusyBox 会判断所调用的是哪个工具，并执行这个工具的功能。我们在这个目录中创建了几个链接来支持 init脚本（每个命令都是一个指向 BusyBox 的链接。）

下一个步骤是创建几个特殊的设备文件。我从自己当前的 /dev 子目录中直接拷贝了这些文件，这使用了 -a 选项（归档）来保留它们的属性。

倒数第二个步骤是生成linuxrc 文件。在内核挂载 RAM磁盘之后，它会查找init 文件来执行。如果没有找到 init 文件，内核就会调用linuxrc文件作为自己的启动脚本。我们在这个文件中实现对环境的基本设置，例如挂载 /proc文件系统。除了 /proc 之外，我还挂载了 /sys文件系统，并向终端打印一条消息。最后，我们调用了 ash（一个 Bourne Shell的克隆），这样就可以与根文件系统进行交互了。linuxrc 文件然后使用 chmod 命令修改成可执行的。

最后，我们的根文件系统就完成了。我们将其卸载掉，然后使用 gzip 对其进行压缩。所生成的文件（ramdisk.img.gz）被拷贝到 /boot 子目录中，这样就可以通过 GNU GRUB 对其进行加载了。

要构建初始 RAM磁盘，我们可以简单地调用 mkird，这样就会自动创建这个映像文件，并将其拷贝到 /boot 目录中。

测试定制的初始 RAM 磁盘

Linux内核中对 initrd 的支持

对于 Linux内核来说，要支持初始 RAM磁盘，内核必须要使用 CONFIG\_BLK\_DEV\_RAM 和 CONFIG\_BLK\_DEV\_INITRD 选项进行编译。

新的 initrd 映像现在已经在 /boot目录中了，因此下一个步骤是使用默认的内核来对其进行测试。现在我们可以重新启动 Linux 系统了。在出现 GRUB 界面时，按 C 键启动GRUB 中的命令行工具。我们现在可以与 GRUB 进行交互，从而定义要加载哪个内核和 initrd映像文件。kernel命令让我们可以指定内核文件，initrd 命令可以用来指定 initrd 映像文件。在定义好这些参数之后，就可以使用 boot命令来引导内核了，如清单 5 所示。

清单 5. 使用 GRUB 手工引导内核和 initrd

GNU GRUB version 0.95 (638K lower / 97216K upper memory)

[ Minimal BASH-like line editing is supported. For the first word, TAB

lists possible command completions. Anywhere else TAB lists the possible

completions of a device/filename. ESC at any time exits.]

grub> kernel /bzImage-2.6.1

[Linux-bzImage, setup=0x1400, size=0x29672e]

grub> initrd /ramdisk.img.gz

[Linux-initrd @ 0x5[f2a](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7568030&ss_c=ssc.citiao.link)000, 0xb5108 bytes]

grub> boot

Uncompressing Linux... OK, booting the kernel.

在内核启动之后，它会检查是否有 initrd映像文件可用（稍后会更详细介绍），然后将其加载，并将其挂载成根文件系统。在清单 6 中我们可以看到这个 Linux启动过程最后的样子。在启动之后，ash shell 就可以用来输入命令了。在这个例子中，我们将浏览一下根文件系统的内容，并查看一下虚拟proc 文件系统中的内容。我们还展示了如何通过 touch 命令在文件系统中创建文件。注意所创建的第一个进程是linuxrc（通常都是init）。

清单 6. 使用简单的 initrd 引导 Linux 内核

...

md: Autodetecting RAID arrays

md: autorun

md: ... autorun DONE.

RAMDISK: Compressed image found at block 0

VFS: Mounted root (ext2 file system).

Freeing unused kernel memory: 208k freed

/ $ ls

bin etclinuxrc proc sys

dev lib lost+found sbin

/ $ cat /proc/1/cmdline

/bin/ash/linuxrc

/ $ cd bin

/bin $ ls

ash cat echo mount sysctl

busybox dmesg ls ps

/bin $ touch zfile

/bin $ ls

ash cat echo mount sysctl

busybox dmesg ls ps zfile

使用初始 RAM 磁盘来引导系统

现在我们已经了解了如何构建并使用定制的初始 RAM磁盘，本节将探索内核是如何识别 initrd 并将其作为根文件系统进行挂载的。我们将介绍启动链中的几个主要函数，并解释一下到底在进行什么操作。

引导加载程序，例如 GRUB，定义了要加载的内核，并将这个内核映像以及相关的 initrd 拷贝到内存中。我们可以在 [Linux内核源代码](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=75978329&ss_c=ssc.citiao.link)目录中的 ./init 子目录中找到很多这种功能。

在内核和 initrd映像被解压并拷贝到内存中之后，内核就会被调用了。它会执行不同的初始化操作，最终您会发现自己到了init/main.c:init()（subdir/file:function）函数中。这个函数执行了大量的子系统初始化操作。此处会执行一个对init/do\_mounts.c:prepare\_[namespace](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=292803&ss_c=ssc.citiao.link)() 的调用，这个函数用来准备名称空间（挂载 dev文件系统、RAID或 md、设备以及最后的 initrd）。加载 initrd 是通过调用init/do\_mounts\_initrd.c:initrd\_load() 实现的。

initrd\_load()函数调用了init/do\_mounts\_rd.c:rd\_load\_image()，它通过调用init/do\_mounts\_rd.c:identify\_ramdisk\_image() 来确定要加载哪个 RAM磁盘。这个函数会检查映像文件的 magic 号来确定它是 minux、etc2、romfs、cramfs 或 gzip 格式。在返回到initrd\_load\_image 之前，它还会调用 init/do\_mounts\_rd:crd\_load()。这个函数负责为 RAM磁盘分配空间，并计算[循环冗余校验码](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=126791&ss_c=ssc.citiao.link" \t "_blank)（CRC），然后对 RAM磁盘映像进行解压，并将其加载到内存中。现在，我们在一个适合挂载的块设备中就有了这个 initrd 映像。

现在使用一个 init/do\_mounts.c:mount\_root()调用将这个块设备挂载到根文件系统上。它会创建根设备，并调用 init/do\_mounts.c:mount\_block\_root()。在这里调用init/do\_mounts.c:do\_mount\_root()，后者又会调用 fs/namespace.c:sys\_mount()来真正挂载根文件系统，然后 chdir 到这个文件系统中。[这就是我们](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=124526757&ss_c=ssc.citiao.link" \t "_blank)在清单 6 中所看到的熟悉消息 VFS: Mounted root(ext2 file system). 的地方。

最后，返回到 init 函数中，并调用init/main.c:run\_init\_process。这会导致调用 execve 来启动 init 进程（在本例中是/linuxrc）。linuxrc 可以是一个可执行程序，也可以是一个脚本（条件是它有脚本解释器可用）。

这些函数的调用层次结构如清单 7 所示。尽管此处并没有列出拷贝和挂载初始 RAM磁盘所涉及的所有函数，但是这足以为我们提供一个整体流程的粗略框架。

清单 7. initrd 加载和挂载过程中所使用的主要函数的层次结构

init/main.c:init

init/do\_mounts.c:prepare\_namespace

init/do\_mounts\_initrd.c:initrd\_load

init/do\_mounts\_rd.c:rd\_load\_image

init/do\_mounts\_rd.c:identify\_ramdisk\_image

init/do\_mounts\_rd.c:crd\_load

lib/inflate.c:gunzip

init/do\_mounts.c:mount\_root

init/do\_mounts.c:mount\_block\_root

init/do\_mounts.c:do\_mount\_root

fs/namespace.c:sys\_mount

init/main.c:run\_init\_process

execve

无盘引导

与嵌入式引导的情况类似，[本地磁盘](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7544279&ss_c=ssc.citiao.link)（软盘或 [CD-ROM](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=31558&ss_c=ssc.citiao.link)）对于引导内核和 ramdisk根文件系统来说都不是必需的。DHCP（Dynamic Host Configuration Protocol）可以用来确定网络参数，例如 IP地址和子网掩码。TFTP（Trivial File Transfer Protocol）可以用来将内核映像和初始ramdisk映像传输到本地设备上。传输完成之后，就可以引导 Linux内核并挂载 initrd 了，这与本地映像引导的过程类似。

压缩 initrd

在构建嵌入式系统时，我们可能希望将 initrd映像文件做得尽可能小，这其中有一些技巧需要考虑。首先是使用 BusyBox（本文中已经展示过了）。BusyBox 可以将数 MB 的工具压缩成几百 KB。

在这个例子中，BusyBox映像是[静态链接](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10727064&ss_c=ssc.citiao.link" \t "_blank)的，因此它不需要其他库。然而，如果我们需要标准的 C 库（我们自己定制的二进制可能需要这个库），除了巨大的 glibc之外，我们还有其他选择。第一个较小的库是 uClibc，这是为对空间要求非常严格的系统准备的一个标准 C 库。另外一个适合空间紧张的环境的库是dietlib。要记住我们需要使用这些库来重新编译想在嵌入式系统中重新编译的二进制文件，因此这需要额外再做一些工作（但是这是非常值得的）。

结束语

初始 RAM磁盘最初是设计用来通过一个临时根文件系统来作为内核到最终的根文件系统之间的桥梁。initrd 对于在嵌入式系统中加载到 RAM磁盘里的非持久性根文件系统来说也非常有用