Better loopback mounts with NBD

Richard W.M. Jones Red Hat Inc. rjones@redhat.com

February 2019

Abstract

Loopback mounts let you mount a raw file as a device. Network Block Device with the nbdkit server takes this concept to the next level. You can mount compressed files. Create block devices from concatenated files. Mount esoteric formats like VMDK. NBD can also be used for testing: You can create giant devices for testing. Inject errors on demand into your block devices to test error detection and recovery. Add delays to make disks deliberately slow. I will also show you how to write block devices using shell scripts, and do advanced visualization of how the kernel and filesystems use block devices.

1 Network Block Device

In the talk there will be an introduction to and history of Network Block Device. I'm not reproducing that here since you can read about the history in articles such as https://www.linuxjournal.com/article/3778. There will also be a short introduction to nbdkit, our pluggable, scriptable NBD server. For now, see https://github.com/libguestfs/nbdkit.

2 Loopback mounts – simple but very limited

Loopback mounting a file is simple:

```
# truncate -s 10M /tmp/test.img
# mke2fs -t ext2 /tmp/test.img
# losetup -f /tmp/test.img
# blockdev --getsize64 /dev/loop0
```

```
10485760
# mount /dev/loop0 /mnt
```

But this talk is about all the things you *cannot* do with a loopback mount. What if the file you want to mount is compressed? What if you want to concatenate several files? What if you want to use another type of storage instead of a file?

You can't do those things with a loopback mount, but there is now an alternative: A loopback Network Block Device, backed by our pluggable, scriptable **nbdkit** server. It's just as simple to use as loopback mounts, but far more flexible.

3 Preparation

If you want to follow these examples on your own machine, you will need to install the `nbd-client` package (on Fedora: `nbd`), and the `nbdkit` server. Most examples require nbdkit $\geq$ 1.7.3.

Linux Network Block Device is in general very reliable, but there were unfortunately a couple bugs in the latest released version that is present in several Linux distributions (but fixed upstream).

If your Linux distro ships with NBD 3.17, make sure it includes the following post-3.17 fix for kernel timeouts: https://github.com/NetworkBlockDevice/nbd/pull/82

If your Linux distro uses kernel $<$ 4.17, then upgrading to 4.17 or above is recommended.

You may also need to run this command once before you start:

```
# modprobe nbd
```

4 Mounting xz-compressed disks

Loopback mounting a compressed disk will expose a block device containing the compressed data, which is not very useful.

nbdkit has a couple of plugins for handling gzip and xz compressed disks. The xz plugin is quite efficient, allowing read-only random access to compressed

files:

```
# nbdkit xz fedora-26.xz
```

We can make a loopback mount called `/dev/nbd0` using one command:

```
# nbd-client -b 512 localhost 10809 /dev/nbd0
```

Linux automatically creates block devices for each partition in the original (Fedora 26) disk image:

```
# ll /dev/nbd0<tab>
nbd0    nbd0p1  nbd0p2  nbd0p3
# file -bsL /dev/nbd0p3
SGI XFS filesystem data (blksz 4096, inosz 512, v2 dirs)
# mount /dev/nbd0p3 /mnt
mount: /mnt: WARNING: device write-protected, mounted read-only.
# cat /mnt/etc/redhat-release
Fedora release 26 (Twenty Six)
```

To clean up:

```
# umount /mnt
# nbd-client -d /dev/nbd0
# killall nbdkit
```

5 Creating a huge btrfs filesystem in memory

nbdkit is not limited to serving files or even to the limits of disk space. You can create enormous filesystems in memory:

```
# nbdkit memory size=$(( 2**63 - 1 ))
# nbd-client -b 512 localhost 10809 /dev/nbd0
```

How big is this? $2^{63} - 1$ is about 8.5 billion gigabytes. If you were to buy that amount of disk at retail it would cost you **€ 300 million**[1].

We can partition and create a filesystem just like any other device:

```
# gdisk /dev/nbd0
Number  Start (sector)    End (sector)  Size       Code  Name
   1            1024  9007199254740973   8.0 EiB     8300  Linux filesystem
# mkfs.btrfs -K /dev/nbd0p1
```

[1] September 2018 prices, WD Red SATA drives bought on Amazon.fr

```
# mount /dev/nbd0p1 /mnt
]# df -h /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/nbd0p1     8.0E   17M  8.0E   1% /mnt
```

When you unmount the NBD partition and kill nbdkit, the device is gone, making this very useful for testing filesystems.

6 Concatenating files into a partitioned disk

Whereas loopback mounts are limited to a single file, there are several nbdkit plugins for combining files. One of them is called the "partitioning" plugin, and it turns partitions into disk images:

```
$ nbdkit partitioning \
         boot.img \
         swap.img \
         root.img
```

This time I'll use `guestfish` to examine this virtual disk:

```
$ guestfish --format=raw -a nbd://localhost -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Fedora 26 (Twenty Six)
/dev/sda3 mounted on /
/dev/sda1 mounted on /boot

><fs> list-filesystems
/dev/sda1: ext4
/dev/sda2: swap
/dev/sda3: xfs
```

You can see that the NBD disk contains three partitions[2].

[2] `/dev/sdX` inside libguestfs is equivalent to `/dev/nbd0` on the host

7 Mounting a VMware VMDK file

VMware VMDK disk images are difficult to open on Linux machines. VMware provides a proprietary library to handle them, and nbdkit has a plugin to handle this library (the plugin is free software, but the VMware library that it talks to is definitely not). We can use this to loopback mount VMDK files:

```
# nbdkit vddk file=TestLinux-disk1.vmdk
# nbd-client -b 512 localhost 10809 /dev/nbd0
```

This disk image contains two partitions and several logical volumes. The Linux kernel finds them all automatically:

```
# file -bsL /dev/nbd0p1
Linux rev 1.0 ext4 filesystem data, UUID=9d1d5cb7-b453-48ac-b83b-76831398232f (n
# file -bsL /dev/nbd0p2
LVM2 PV (Linux Logical Volume Manager), UUID: bIY2oM-CgAN-npqG-gItS-WY6e-wO7d-L6
# ls /dev/vg_testlinux/
lv_root  lv_swap
```

You can read and write to VMDK files this way:

```
# mount /dev/vg_testlinux/lv_root /mnt
# touch /mnt/hello
```

8 Testing a RAID array

Let's make a RAID array using in-memory block devices. But to test them we'll want a way to inject errors into those block devices. nbdkit makes this easy with its *error filter*:

```
# nbdkit --filter=error memory size=1G \
         error-file=/tmp/error0 error-rate=1 -p 10810
# nbdkit --filter=error memory size=1G \
         error-file=/tmp/error1 error-rate=1 -p 10811
# nbdkit --filter=error memory size=1G \
         error-file=/tmp/error2 error-rate=1 -p 10812
# nbdkit --filter=error memory size=1G \
         error-file=/tmp/error3 error-rate=1 -p 10813
# nbdkit --filter=error memory size=1G \
         error-file=/tmp/error4 error-rate=1 -p 10814
# nbdkit --filter=error memory size=1G \
```

```
        error-file=/tmp/error5 error-rate=1 -p 10815
```

We can create 6 NBD devices from these:

```
# nbd-client localhost 10810 /dev/nbd0
# nbd-client localhost 10811 /dev/nbd1
# nbd-client localhost 10812 /dev/nbd2
# nbd-client localhost 10813 /dev/nbd3
# nbd-client localhost 10814 /dev/nbd4
# nbd-client localhost 10815 /dev/nbd5
```

And we can create a RAID 5 device on top:

```
# mdadm -C /dev/md0 --level=5 \
        --raid-devices=5 --spare-devices=1 \
        /dev/nbd{0,1,2,3,4,5}
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
# mkfs -t ext4 /dev/md0
# mount /dev/md0 /mnt
```

You can see we have 5 drives and 1 spare in the array:

```
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 nbd4[6] nbd5[5](S) nbd3[3] nbd2[2] nbd1[1] nbd0[0]
      4186112 blocks super 1.2 level 5, 512k chunk, algorithm 2 [5/5] [UUUUU]
```

nbdkit's error filter is trigger by the presence of the error files `/tmp/error*`. By creating these files we can inject errors into specific devices and see how the RAID array responds.

Firstly I inject errors into `/dev/nbd0`:

```
# touch /tmp/error0
```

After a while the kernel notices:

```
[10804.798999] print_req_error: I/O error, dev nbd0, sector 100360
[10804.868378] md: recovery of RAID array md0
[10805.202631] md/raid:md0: read error corrected (8 sectors at 69928 on nbd0)
[10810.349550] md: md0: recovery done.
```

Comparing `/proc/mdstat` before and after:

```
-md0 : active raid5 nbd4[6] nbd5[5](S) nbd3[3] nbd2[2] nbd1[1] nbd0[0]
+md0 : active raid5 nbd4[6] nbd5[5] nbd3[3] nbd2[2] nbd1[1] nbd0[0](F)
```

shows that the spare drive is now in use and nbd0 is marked as Failed.

I can inject errors into a second drive:

```
# touch /tmp/error1
[11039.428009] block nbd1: Other side returned error (5)
[11039.431659] print_req_error: I/O error, dev nbd1, sector 231424
[11039.448757] block nbd1: Other side returned error (5)
[11039.452367] print_req_error: I/O error, dev nbd1, sector 233280
[11084.767968] md/raid:md0: Disk failure on nbd1, disabling device.
               md/raid:md0: Operation continuing on 4 devices.
```

and now the array is operating in a degraded state. At the filesystem level everything is still fine.

9 Writing a Linux block device in shell script

nbdkit allows you to write plugins in various programming languages, including shell script. In the talk I will demonstrate a Linux block device being written as a shell script.

10 Logging and visualization

I am planning some visualization tools that will let you see exactly how a block device is being read and written during common operations like filesystem creation, file allocation, fstrim, and so on. The talk will end with a demonstration of these tools.