ECE 590-03: Enterprise Storage
Fall 2016
Homework 2

Last updated 2016-10-01

While Homework 1 consisted of primarily pen-and-paper theory questions, this homework is almost entirely practical experiments with storage. To do it, you'll need access to two systems:

- A Linux-based **storage controller**, a Linux machine on which you have root access. This could be a Duke OIT VM, a VM you install yourself on your computer, a physical computer you own, an Amazon EC2 instance[1], or another Linux VM hosting service. The login service at login.oit.duke.edu is not acceptable for this role, as you do not have root access on those computers. The Linux version should be fairly recent, and support NTFS, such as Ubuntu 14.04 or later (other distros, such as Arch or Fedora, will also work with minor adjustments to some steps).
- A **Windows client** (Windows 7, 8, or 10) for which you have administrator access. If your personal computer is Windows-based, you can use it. If you run Mac/Linux on your personal computer, set up a Windows VM in a free hypervisor, such as VirtualBox. The Windows client will need network access to the storage controller.

This homework will consist of a series of installation and administration tasks to set up a fairly decent storage environment using free software. To be clear, in industry, a decent fraction of storage is NOT based on free software, but all the fundamental principles are the same, and if you can build this from scratch in Linux, you should be able to administer a fluffy GUI-based storage solution without issue.

This homework assumes some basic command-line proficiency in Linux. If your background is lacking in that area, you'll need to do a bit of research. You can always ask Dr. Bletsch if you get stuck.

To provide "answers" to this homework, you are asked to provide screenshot(s) or terminal logs. If there a lot of extraneous commands or output, you can edit it out, or just highlight or mark the relevant commands so we can spot them easily. Make sure you don't edit out any necessary steps.

The actual format for submission should be a PDF with clear labels for tasks and steps, with screenshots or terminal logs pasted inline. Please format this document for ease of reading – grading will be based on reading through the output to ensure correct functionality.

---

[1] If you use Amazon for this, note that the default security settings block incoming traffic, so exceptions will need to be made to allow CIFS and iSCSI access.

Task 1: Preparing fake hard drives

We'll assume your storage controller Linux system has a basic install and is providing SSH login access. If not, make it so.

The first thing we're going to do is RAID, but your storage controller likely does not have multiple physical drives. Therefore, we're going to fake it. Become root ("sudo su -"), then, following the example here, create four 100MB files: `/root/fakedisk0`, `/root/fakedisk1`, `/root/fakedisk2`, and `/root/fakedisk3`. When done, you should see them like this:

```
root@xub1404dt:~ # pwd
/root
root@xub1404dt:~ # ls -l fakedisk*
-rw-r--r-- 1 root root 104857600 Sep 30 15:52 fakedisk0
-rw-r--r-- 1 root root 104857600 Sep 30 15:52 fakedisk1
-rw-r--r-- 1 root root 104857600 Sep 30 15:52 fakedisk2
-rw-r--r-- 1 root root 104857600 Sep 30 15:52 fakedisk3
```

Next, we need to tell the Linux kernel to treat these four regular files as if they were actual block devices. Linux has support for this in the form of "loop devices". Use the losetup utility to map each of our fake disk files to `/dev/loop0` through `/dev/loop3`. You can verify that the loop devices are recognized with the losetup list command and by checking the kernel partitions status file `/proc/partitions`, e.g.:

```
root@xub1404dt:~/.ssh # losetup -a
/dev/loop0: [0801]:794211 (/root/fakedisk0)
/dev/loop1: [0801]:797548 (/root/fakedisk1)
/dev/loop2: [0801]:797549 (/root/fakedisk2)
/dev/loop3: [0801]:797550 (/root/fakedisk3)

root@xub1404dt:~ # cat /proc/partitions
major minor  #blocks  name

   7        0     102400 loop0
   7        1     102400 loop1
   7        2     102400 loop2
   7        3     102400 loop3
  11        0    1048575 sr0
   8        0   20971520 sda
   8        1   19921920 sda1
   8        2          1 sda2
   8        5    1046528 sda5
```

Provide screenshots or a terminal log of these steps in a section marked "Task 1".

Task 2: Set up software RAID

We'll be using the Linux "md" subsystem to set up software RAID. Ensure the "mdadm" tool is installed.

Noting this RAID setup guide, create a three-disk RAID5 called `/dev/md0` using devices /dev/loop0 through /dev/loop2; also include /dev/loop3 as a spare device. You can always check the current status of all md RAID devices by reading /proc/mdstat. When I built mine, I checked this file immediately after to see it "repairing" the new array (calculating and storing parity). This took about one second. In this output, you can see mdstat showing the rebuild underway, and then later showing the array as ready:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop2[4] loop3[3](S) loop1[1] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2]
[UU_]
      [================>....]  recovery = 80.5% (82836/101888)
finish=0.0min speed=27612K/sec

unused devices: <none>
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop2[4] loop3[3](S) loop1[1] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

Note the "(S)" flag on loop3: this indicates that this device is a hot spare.

Provide screenshots or a terminal log of these steps in a section marked "Task 2".

Task 3: Set up a file system

Our RAID array is now represented by `/dev/md0`. Now we need a file system on top of it. As we learned in class, there are many file systems available, and even a basic install of Linux supports many of them. To create the initial filesystem, we use the mkfs ("make file system") utility, which has variants for each supported filesystem (e.g. ext2, ext3, ntfs, etc.). Later in this assignment, we're going to serve this block device over a SAN to the Windows client, so let's pick the modern Windows filesystem, NTFS. Use "mkfs.ntfs" to prepare a filesystem on `/dev/md0`.

Now we need a **mount point**, a directory into which we can attach the filesystem. Using mkdir, create a directory called "`/x`".

Now use the mount utility to mount our `/dev/md0` filesystem to `/x`. Note that while the mount utility does support a lot of options, none are needed for this basic operation – the tool will have no trouble auto-detecting the type of filesystem present and using reasonable default settings when mounting it. When done, you can verify by typing "mount" by itself to list mounted filesystems; you can also type "df" to check free space. Example:

```
root@xub1404dt:~ # mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
 ...
none on /sys/fs/pstore type pstore (rw)
systemd on /sys/fs/cgroup/systemd type cgroup
(rw,noexec,nosuid,nodev,none,name=systemd)
/dev/md0 on /x type fuseblk (rw,nosuid,nodev,allow_other,blksize=4096)

root@xub1404dt:~ # df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       19478204 4191128  14274596  23% /
none                   4       0         4   0% /sys/fs/cgroup
udev              489424       4    489420   1% /dev
tmpfs             100020    1456     98564   2% /run
none                5120       0      5120   0% /run/lock
none              500096     152    499944   1% /run/shm
none              102400      40    102360   1% /run/user
/dev/md0          203772    2504    201268   2% /x
```

Provide screenshots or a terminal log of these steps in a section marked "Task 3".

Task 4: Use the file system

By whatever means you wish, populate the filesystem at least a few kilobytes of content. At least one file should be a text file you can verify the contents of. When done, you should be able to cd to your filesystem and see your stuff, e.g.:

```
root@xub1404dt:/x # ls -l
total 781
-rwxrwxrwx 1 root root 797772 Sep 30 17:14 kern.log
-rwxrwxrwx 1 root root     21 Sep 30 17:15 test.txt
root@xub1404dt:/x # cat test.txt
This is my text file
```

Provide screenshots or a terminal log of these steps in a section marked "Task 4". Be sure to include a file listing and "cat" of your small text file, as shown above.

Task 5: Damage the RAID

We'll now simulate failure of one of the disks. As soon as you do, the array will begin repairing itself using the hot spare we provided. Swift action will be needed to see the rebuild in progress, since our array is so small.

Use the mdadm command to "fail" /dev/loop1 in the array, then immediately (within one second) print the content of /proc/mdstat so you see the array status while it's rebuilding. Check the /proc/mdstat file again after it's done to see the final status. You'll find the failed disk with a "(F)" flag, and the former spare disk is now in the array, so it no longer has the "(S)" flag. Example output of mdstat before, during, and after rebuilding:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop3[3](S) loop1[5] loop2[4] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]
```

(failure is simulated)

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop3[3] loop1[5](F) loop2[4] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/2]
[U_U]
      [==========>.........]  recovery = 50.0% (51580/101888)
finish=0.0min speed=51580K/sec

unused devices: <none>
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop3[3] loop1[5](F) loop2[4] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

Verify that the content in the mounted filesystem `/x` is undamaged, despite a disk failure, e.g.:

```
root@xub1404dt:/x # ls -l
total 781
-rwxrwxrwx 1 root root 797772 Sep 30 17:14 kern.log
-rwxrwxrwx 1 root root     21 Sep 30 17:15 test.txt
root@xub1404dt:/x # cat test.txt
This is my text file
```

Provide screenshots or a terminal log of these steps in a section marked "Task 5".

Task 6: Replace the faulty drive

At this point, because the spare was there, your array can again withstand a drive failure, but there would be no more spares to rebuild it immediately in that case. Running with no spares is dangerous, so we'll now simulate replacing the faulty drive. Steps:

1. Use the mdadm "remove" option to delete the faulty drive /dev/loop1 from of the array listing.
2. Use losetup to detach /dev/loop1 from /root/fakedisk1.
3. Use dd to create a new 100MB file /root/newdisk1
4. Use losetup to tie that new file to /dev/loop1
5. Use the mdadm "add" option to put this new blank disk in our `/dev/md0` array

When you're done, /dev/loop1 should show as a new spare device in /proc/mdstat:

```
root@xub1404dt:~ # cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 loop1[5](S) loop3[3] loop2[4] loop0[0]
      203776 blocks super 1.2 level 5, 512k chunk, algorithm 2 [3/3]
[UUU]

unused devices: <none>
```

Your array is now capable of immediately rebuilding if another drive failure occurs.

Provide screenshots or a terminal log of these steps in a section marked "Task 6".

Task 7: Deploy a CIFS NAS

Install the samba server package, which serves CIFS shares compatible with Microsoft Windows clients. Configure it to share our filesystem in `/x` as a share called "x". There are a lot of tutorials on this, so I leave the details to you.

To test it, login to your **Windows client**. In Windows Explorer, navigate to "\\<*IP_ADDRESS_OF_STORAGE_CONTROLLER*>\x". The files you placed there in Task 4 should be visible.

Provide screenshots or a terminal log of these steps in a section marked "Task 7".

IN ADDITION, provide the relevant portion of your smb.conf file and a screenshot of Windows Explorer showing your storage controller NAS share.

Task 8: Create an iSCSI SAN

Now, instead of a CIFS NAS, we're going to instead deploy an iSCSI SAN. However, we cannot serve the filesystem `/x` and the block device `/dev/md0` at the same time, since `/x` resides on `/dev/md0`. If we did, both the Linux storage controller and the Windows SAN client would be making changes to `/dev/md0`, with neither aware of the other. This would lead to file system corruption and inconsistency.

Therefore, begin by stopping the samba service and using `umount` to unmount `/x`. At this point, `/dev/md0` should not appear in your mount list:

```
root@xub1404dt:/ # mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
 ...
systemd on /sys/fs/cgroup/systemd type cgroup
(rw,noexec,nosuid,nodev,none,name=systemd)
```

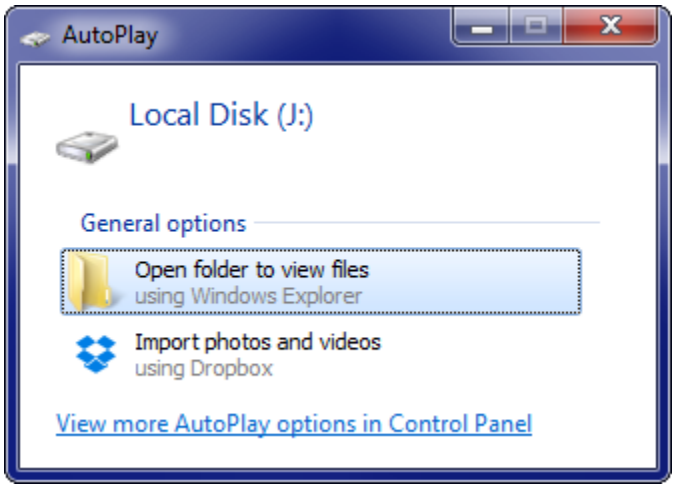
(no /dev/md0 in this list)

To set up the iSCSI target, we can use this guide[2] as a base, but with some differences:

- The guide asks you to use dd to create a dummy file to be our LUN; we're going to use our `/dev/md0` device instead.
- When you get to setting up your LUNs in ietd.conf, set your storage controller's IQN to "`iqn.1986-06.edu.duke:<YOUR_NET_ID>`".
- Your LUN path will be `/dev/md0` (your RAID block device).
- The LUN "Type" should be set to "blockio" rather than "fileio".

[2] This guide is based on Ubuntu or Debian based Linux distributions. If you're using another distribution of Linux, just find an appropriate guide. Basically, the steps to install and enable the target will be different, but the rest should be the similar.

Our block device already has a Windows-compatible NTFS file system on it, so we're going to use our Windows client to connect to it. As described in the guide, open the Windows iSCSI Initiator and point it to the IP address of your storage controller. Connect it and attach the LUN, and if all goes well, Windows should not only detect the block device, but Explorer should pop up to ask you what you want to do with your new drive:



When you view the new drive, you should see the content you put there in Task 4. So we have access to the same data as with the NAS, but now Windows is the one doing the file system logic, so the data shows up as its own drive, and Windows is sending simple read-block/write-block requests to the SAN target.

Provide screenshots or a terminal log of these steps in a section marked "Task 8".

IN ADDITION, provide the relevant portion of your ietd.conf file, a screenshot of the Windows iSCSI Initiator Properties window, and a screenshot of Windows Explorer showing your attached SAN drive.

Task 9: Backups with rsnapshot

Let's use our storage controller to build an automated backup system. Normally, we'd be backing up data from another server, but I don't want to ask you to set up yet another Linux machine, and the Duke Linux login machines don't support passwordless login, so we're just going to configure local backup within the storage controller.

You can use this guide[3], and note the following:

- **Backup source**: We're going to back up your user home directory on the VM. This is `/home/<USERNAME>`, where <USERNAME> is whatever user you created when you installed Linux. (If you somehow installed Linux without creating a non-root user, make a user account now, login as the user, and put some stuff in its home directory.)
- **Backup destination:** We need to use a native Linux filesystem for this, because rsnapshot depends on hard links, which are not supported in NTFS. Therefore, we can ignore the software RAID stuff we just did, and instead make a directory `/backup` as your destination (so data will be stored in the root partition of the VM).
- **Frequency:** Set up nightly and weekly intervals.
- **Automation:** You do NOT need to do cron-based automation. Once you have rsnapshot configured, you can just run "`rsnapshot nightly`" to perform the backup that would happen nightly, and "`rsnapshot weekly`" to perform the backup that would happen weekly. You can run these commands as often as you wish; there's nothing in the software that actually needs the backups to be done nightly/weekly as opposed to every few dozen seconds (i.e., rsnapshot does not actually care that they're called "nightly" or "weekly").

Open one terminal window as root, and another terminal window as the non-root user. Do the following, **and for each step, show the process via screenshots or terminal logs. Label or otherwise identify each step you're doing in your screenshot/log.**

1. As the user, add some content to the user home directory.
2. As root, do several nightly and weekly backups to populate your backups.
    - Observe how the hard-link count changes for files as you take more and more backups.
    - Check the disk free space on the root partition as you take backups; the storage increase should be very small (just metadata).
3. As the user, delete an important file.
4. As root, take another few nightly backups to simulate time passing.
5. As the user, you should still have read access to the backups. Use this access to copy your deleted file from an older nightly backup to your home directory, thus restoring the file.

Provide screenshots or a terminal log of these steps in a section marked "Task 9".

IN ADDITION, provide the relevant portions of rsnapshot.conf.

[3] The installation step is distro-specific, but everything else is distro-neutral.