Problem 1 -- Filesystem data structures

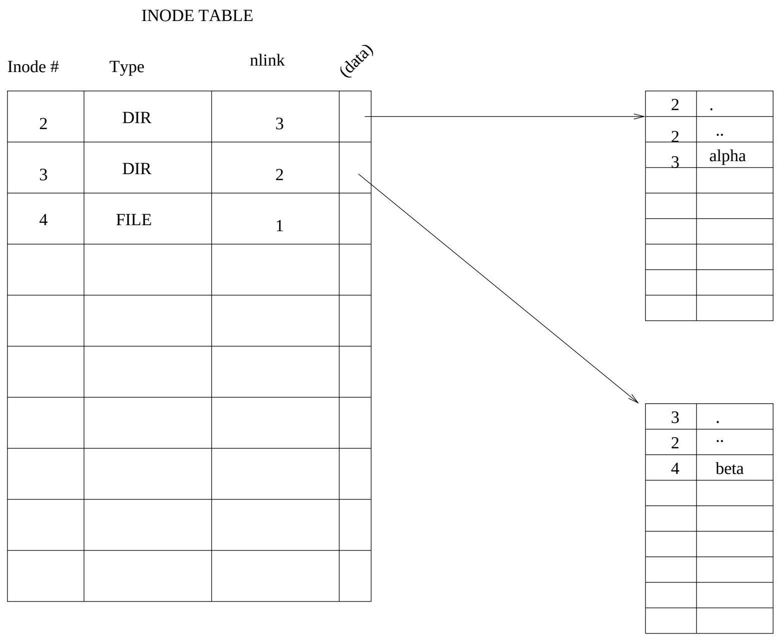
In this problem, you'll trace out the operation of a simple, hypothetical filesystem which follows the principles laid out under "UNIX Filesystems" in the lecture notes. Specifically, we envision this filesystem as having a superblock, inode table, free map and data block areas. To simplify this problem, we can represent the inode "table" schematically, as in the lecture notes, as a table with rows representing inodes and columns the fields of the inode. I.e. we won't concern ourselves with the actual on-disk layout of the inode. We'll also ignore the superblock and the free map. We shall not worry about how the filesystem actually allocates blocks. Directories will be represented as a table of name/inode# pairs. We won't worry about how the directory is actually laid out, or how much disk space it takes. Here are a few more assumptions:

- The first inode is #2 (as is conventional)
- When allocating an inode, the lowest numbered free inode is used
- Directories are in order of directory entry creation time (not sorted by name or inode #, etc.)

Now, let's get started. I initialized a volume and made the first file and the first subdirectory:

```
mkfs /dev/sda1    #Don't try this at home, folks!
mount /dev/sda1 /mnt1
mkdir /mnt1/alpha
echo "TEST123" >/mnt1/alpha/beta
```

This is what the volume looks like at this point, so you have a clear idea of the representational format:



Now I perform the following operations:

```
mkdir /mnt1/alpha/gamma
ln /mnt1/alpha/beta /mnt1/alpha/gamma/delta
ln -s /mnt1/alpha/beta /mnt1/epsilon
rm /mnt1/alpha/beta
echo "ggg" >/mnt1/alpha/zeta
```

Sketch the inode table and directories at this point. This diagram must be **NEAT**! Preferably you will create it using your favorite line art or CAD program and export as PDF. If you are submitting hand-drawn work, it must be free of cross-outs, all writing must be legible, and the scan should be clean, in-focus, and properly exposed (i.e. not a blurry cell phone snapshot of the wrinkled page sitting on your kitchen table)

Problem 2 -- Exploratory Questions

Answer each of the following with a paragraph or two **in your own words**. After you read the lecture notes, you'll have enough material to answer.

A) The generic "UNIX" filesystem is depicted on page 5 of the lecture notes with header (superblock), inode table, free map, and data block areas. On page 31 there is a more literal depiction of the Linux EXT3 filesystem showing a "cylinder group" type of organization where these areas are distributed across the volume. Why is this advantageous for mechanical hard disks but of little value for SSDs?

B) I have an EXT4 volume (or any "UNIX" type filesystem) and I explore the inode table. A particular inode, let's say inode #6688, has a `mode` field of 040755 (this is an octal number) and `nlink` is 5.

i) What type of inode is this?

ii) Without exploring anything else, what inference can we draw from the `nlink` value?

C) A user purchases and installs a 4TB SATA disk to hold a collection of 4K-quality vids and installs it on a desktop PC running Linux. Each video file is exactly 2^30 bytes long. Of course, this user follows good security practices and does not download or watch the videos as "root." The expectation was that approx. 4000+ videos would fit on this disk, but the reality was noticeably less. Give at least two reasons why this would be the case.

D) List at least 3 reasons why a `umount` command would fail

E) I run the command `ls -l` in a directory that has tens of thousands of entries. The first time I run this command, it takes several seconds to generate its output. But when I run it again a short time later, it is much, much quicker. Explain which fileystem caches have improved performance, how, and why.

Problem 3 -- Recursive filesystem lister

Write a program in C which uses the directory exploration library functions such as `readdir` to produce a recursive directory listing which is similar to the output of `find /starting_directory -ls`

The program takes one optional argument which is the starting directory. If this is omitted, it defaults to "./" which is the current directory. Output of the program goes to stdout. Any errors must go to stderr, not stdout! There is no prohibition against using library functions, except that you must use opendir(3) and related functions to scan directories. Don't cheat by using something like ftw(3)

First, make some test cases by making nested directories with various files inside. See what the `find` command does. This will give you a basis of comparison with your own program output. If you format your output the same way, you can even use the `diff` command to compare. On the course web site, you'll find a small sample filesystem which can be used for testing. You can mount this with the following command, as root:

```
mount -oloop ./testfs.img /mnt
```

You should also run your own tests, e.g. on your own Linux system's file tree.

For reference, the output of `find` with the `-ls` option is:

Inode #, disk usage of file (measured in units of 1K), inode type and mode string, link count, uid name (or number if no number->name translation is found), gid name (ditto), file size, mtime in human-readable format, full path name, for symlinks only: the string " -> " and the contents of the symlink

This output is identical to what would be produced by running `ls -dils` on each full path name.

Note that the order in which the directory entries are printed is not necessarily sorted. It is simply the order in which `readdir` returns the entries. Along with that, the order of recursion (e.g. depth-first) is not defined. Subdirectories are recursively descended as they are encountered (as opposed to, for example, exploring all the subdirectories first before listing the contents of the current directory).

There are some issues which we'll consider beyond the scope of this simple program. 1) you aren't required to detect loops in the filesystem tree, which after all could only happen if someone managed to create a forbidden hard link to a directory 2) you don't have to worry about running out of file descriptors for opendir because of a tree that is too deep. Just detect the error in this case, but no need to recover from it.

Gotchas: 1) The translation from numeric uid and gid to a name depends on the system password database. If you have a volume which came from another system, it is possible (even likely) that some uid/gid numbers will have no valid lookup on your system. This is not a fatal error, nor should it cause your program to crash. 2) You should try to emulate as closely as possible the mode string that `ls` would output. If you encounter an inode type that is not defined, output a ? in the first column, and/or print an error message. 3) Any date output format is acceptable, note that the UNIX `ls` command outputs local time (whereas the timestamp is actually stored in the inode in UTC) and has a few different formats depending on how recent the timestamp is.

Do not use the `strmode` function to decode the mode (rwx, etc.) Write this yourself!

Submission: the program source code, some screenshots showing the program running. If you used diff to verify your output, show that too.

Problem 4 -- Extra Credit Filesystem corruption and recovery-- 1pt

Attach screenshots (or terminal session text pastes) as you perform the following system maintenance tasks. Note: you must do these as **root** (uid 0) on a real UNIX system. You can't do this extra credit part on the ece357.cooper.edu VM. The examples below are specific to Linux. Be very careful since an error in one of these commands could destroy data on your hard disk! There are numerous online resources to help you with this process.

1) Obtain a USB memory stick or USB external hard disk that you don't care about.

2) Attach this device to your system. Using the `dmesg` command you should see some system log messages which reveal which device node the kernel has assigned to the drive. A typical series of log messages is below:

```
[1309116.520803] usb 1-4: new high-speed USB device number 3 using ehci-pci
```

```
[1309116.648472] usb 1-4: New USB device found, idVendor=154b, idProduct=005b
[1309116.648475] usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[1309116.648477] usb 1-4: Product: USB 2.0 FD
[1309116.648479] usb 1-4: Manufacturer: PNY Technologies
[1309116.648481] usb 1-4: SerialNumber: AA00000000000033
[1309116.648836] usb-storage 1-4:1.0: USB Mass Storage device detected
[1309116.648947] scsi10 : usb-storage 1-4:1.0
[1309118.001013] scsi 10:0:0:0: Direct-Access     PNY      USB 2.0 FD       1100 PQ: 0 ANSI:
[1309118.001181] sd 10:0:0:0: Attached scsi generic sg8 type 0
[1309118.002042] sd 10:0:0:0: [sdh] 63901440 512-byte logical blocks: (32.7 GB/30.4 GiB)
[1309118.002862] sd 10:0:0:0: [sdh] Write Protect is off
[1309118.002864] sd 10:0:0:0: [sdh] Mode Sense: 43 00 00 00
[1309118.003578] sd 10:0:0:0: [sdh] No Caching mode page found
[1309118.003580] sd 10:0:0:0: [sdh] Assuming drive cache: write through
[1309118.007737]  sdh: sdh1
[1309118.010834] sd 10:0:0:0: [sdh] Attached SCSI removable disk
```

Verify that the description of the device matches what you just inserted! Note that the kernel has assigned `/dev/sdh` to the overall disk, and that there is one partition `/dev/sdh1`.

Another handy command is `lsblk` which will show you all the block devices on your system along with major/minor numbers and size. **Make sure you have the correct device name!** Remember, as root you can destroy your entire system!

3) Make a Linux EXT2 filesystem on the disk (OK to use either the partition or the entire disk). Alternatively, make an EXT3 filesystem, but do not enable journalling. Show the commands and responses.

4) Mount the disk. Show the output of the `mount` commands (without any arguments) which shows the mounted volume (highlight the relevant line). Run a test program to generate endless disk metadata activity, e.g. repeatedly create, rename and delete files. If this test program produces a lot of debugging output you don't need to include that in the screenshots. Include the source code of this program.

5) Disconnect the disk while it is active. CAUTION: this may hang your system for a while, and may require a reboot to recover.

6) Reconnect the disk. Run fsck to repair the filesystem (note, you may have to use `fsck.ext2` or `fsck.ext3` specifically, depending on your distribution, as it may attempt to fsck the disk as a FAT filesystem first). You should see at the very least that the filesystem was "not cleanly unmounted." If you've generated enough corruption from your interrupted activity, you may see other errors as well. Answer "y" to allow repairs to take place. Now mount it and see if any files have appeared in `lost+found`. Show all these commands and outputs. Umount the filesystem. We are done with this first test.

7) Make a new Linux EXT3 or EXT4 filesystem on the disk (overwriting the previous filesystem). Make sure journalling IS enabled this time.

8) Mount the filesystem and perform your steps 4 and 5 again.

9) Reconnect the disk and attempt to mount it. Verify from the logs that the journal was recovered. Show (highlight) the relevant lines of the dmesg logs relating to journal recovery.