This laboratory explores links created in file systems commonly used on UNIX systems.

Links are special references to files and directories. There are two types of links: hard links and symbolic links.

Setup

First, run:

```
% ~bmcdonald/class/lab08.sh
```

Next, run:

```
% cd ~/lab08
```

Finally, run:

```
% ls
```

You should see files named *originalFile* and *linkedFile*.

Hard Links

Hard links create duplicate references to the same file.

Run:

```
% ln originalFile hardlink
```

This creates a hard link named *hardlink*, which links to the file *originalFile*.

Then run

```
% ls -l
```

Notice that all the information on the file and the hard link are identical, except for the name. That is because the file and the hard link point to the same file.

Try:

```
% diff originalFile hardlink
```

There will be no output from *diff*, which is the case when the two files compared are the same.

If you change one, the other one will be changed.

Run:

```
% vi hardlink
```

Add a line to the file by:

1. hitting `o`
2. typing a line of text
3. hitting the Escape key to go back to command mode

4. typing `:wq` to exit *vi*.

Now, cat both the file and the hard link:

```
% cat originalFile
% cat hardlink
```

Notice that the output of both is the same. You've changed the contents of both files by editing only one of the files.

For an automated way of making this comparison, run:

```
% diff originalFile hardlink
```

Again, diff produces no output, indicating that the files referred to by the file system entries are not only identical, they are the same file.

Deleting a Hard Link

Use the *rm* command to delete a hard link, just as you would when deleting a file.

Try:

% rm hardlink

Then:

% ls -l

The file is still there, and will be there unless all references to the file are deleted. That means that you could do the following and retain the file system entity referenced by the file and hard link entities:

```
% ln originalFile hardlink
% diff originalFile hardlink
% rm originalFile
% ls
% cat hardlink
% ln hardlink originalFile
% ls -l
% diff hardlink originalFile
```

This creates a hard link and removes the original file entry, but retains the contents of the file, so that you can recreate the entry for the original file because you haven't deleted the file pointed at by the file system entries.

You could run the above set of commands after substituting the *cp* command for the *ln* command. However, the two entries would not reference the same inode, so that if you edited one, the other one would remain the same. This is the difference between making a hard link to a file and copying the file.

Verifying Hard Linking

You can see if two file system entries are in fact hard links to the same file by running:

```
% ls -il
```

The first column in the listing is for the *inode* of the entry. The inode is the unique identifier for the entity in the file system to which one or more file system entries refer. Notice the two files refer to the same inode since the inode number is the same for both.

Limitations of Using Hard Links

You can't create a hard link that refers to a directory, or if you can, you can create problems.

You can create a hard link only within the same disk or disk partition. Remember what I said about mount points being the place where a given disk or disk partition is attached to the single file system hierarchy supported by UNIX systems. We'll learn more about disk partitioning later.

If you back up or copy a directory that contains two hard links to the same file, you will end up with two copies of that file, which is a waste of space.

The above limitations are why hard links aren't used as much as symbolic links.

Symbolic Links

Symbolic links (often called symlinks) have similar results to hard links, except that symbolic links are not duplicate references to the same inode as is referred to by the original file.

Symbolic links have numerous advantages over hard links:

- you can safely create a symbolic link to a directory
- backups of a directory do not backup multiple copies of a file that is referenced by symlinks
- you can create a symbolic link to a file or directory that is on a different disk or disk partition
- many UNIX utility programs handle symlinks appropriately

Symbolic links are the file system equivalent of what programmers call pointers. That is why they are useful. You can use symbolic links to create the appearance of a seamless file/directory hierarchy even if the hierarchy isn't complete or is changing. This gives you flexibility. It also lets you save space in your file systems, since symlinks just create a small pointer and do not copy the contents of the linked file.

Run:

```
% ln -s linkedFile symlink
```

This creates a symbolic link named *symlink*.

Look at the file system entries:

```
% ls -il
```

Notice that the inodes differ between the symlink and the file pointed at by the symlink.

Run:

```
% ln -s linkedFile symlink2
```

Look at the file system entries:

```
% ls -il
```

Notice that the inodes differ between the two symlinks.

Compare the file with the symlink to the file:

```
% diff linkedFile symlink
```

You should get no output from *diff*.

You can edit the file either directly or through its symlink.

Run:

```
% vi linkedFile
```

Add a line to the file by:

1. hitting `o`
2. typing a line of text
3. hitting the Escape key to go back to command mode
4. typing `:wq` to exit *vi*.

Then:

```
% diff linkedFile symlink
```

The files are identical.

Try:

```
% cat linkedFile
% cat symlink
```

See that the contents are identical.

Try the same thing with the symlink. Run:

```
% vi symlink
```

Add another line, or make some other obvious change, saving the changes and exiting vi.

Again:

```
% diff linkedFile symlink
```

The files are still identical.

Deleting Symlinks

As with hard links, you can remove a symlink using the *rm* command.

```
% rm symlink2
```

```
% ls
```

Notice that the original file is still there after deleting the symlink to it.

Changing the Symlink to Point to a Different File

You can make a symlink refer to a different file by deleting the symlink, then recreating it.

```
% ln -s originalFile symlink2
```

```
% cat symlink2
```

This ability to change what gets pointed at by the symlink is what makes the symlink very valuable. If you set up a file system hierarchy to support other users, you can tell them that the file they need is always at a certain location. If the entry for that location is a symlink, you can change the symlink to point to whatever file is appropriate at the time, without having to tell the users that they must get the current file from a different location. Giving your users a constant location from which to get what they need is much easier on your users than making them keep up with your underlying changes.

Moving the File Referenced by a Symlink

Run:

```
% ln -s linkedFile symlink3
```

```
% cat symlink3
```

The *cat* command dumps the file referred to by the symlink.

Then:

```
% mv linkedFile moved
```

```
% cat symlink3
```

The *cat* command can't dump the file referenced by the symlink.

Replace it with another file:

```
% cp originalFile linkedFile
```

```
% cat symlink3
```

The *cat* command dumps the contents of the file referenced by the symlink.

Creating a Symlink to a Directory

Again, you can create a symlink to a directory. This has the same conveniences as creating symlinks to files, but has the following consequences:

cd'ing Using a Symlink can Give Unexpected Results

If you *cd* into a directory using a symlink that refers to that directory, if you subsequently move upward in the file system hierarchy, you will move upward through the hierarchy at the symlinked directory, not back to the directory containing the symlink.

pwd vs. /bin/pwd

If you *cd* into a directory using a symlink that refers to that directory, *pwd* will report the symlink as the directory. If you want to know the actual path of your current directory, you can run:

```
% /bin/pwd
```

to find out.