Shared (Multinamed) Files (Links)

We often think of the files and directories in a file system as forming a tree (or forest). However in most modern systems this is not necessarily the case, the same file can appear in two different directories (not two copies of the file, but the *same* file). It can also appear multiple times in the same directory, having different names each time.

I like to say that the same file has two different names. One can also think of the file as being shared by the two directories

Note: A file can also have two names in the same directory

- "Shared" files is Tanenbaum's terminology.
- If a file exists, one can create another name for it (quite possibly in another directory). Or in the same directory.
- This is often called creating another link to the file.
- Unix has two flavor of links, **hard links** and **symbolic links** (a.k.a. **symlinks**).
- Dos/windows has shortcuts, which behave somewhat like symlinks
- We will concentrate on both flavors of unix links.
- These links often cause confusion

Windows does not have a 'hardlink' concept

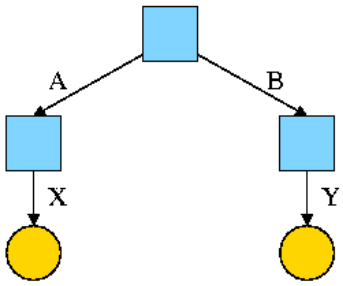
Hard Links

With unix hard links there are multiple names for the same file and each name has equal status. The directory entries for both names point to the **same** inode.

- Hard links are thus *symmetric* multinamed files.
- When a hard link is created *another* name is created for the *same* file. The number of files in the system is *the same* before and after the hard link is created.
- It is *not*, I repeat **NOT**, true that one name is the "real name" and the other one is "just a link".
- Indeed after the hard link has been created it is not possible to tell which was the original name and which is the newly created link.

For example, the diagram on the right illustrates the result that occurs when, starting with an empty file system (i.e., just the root directory) one executes

```
cd /
mkdir /A; mkdir /B
touch /A/X; touch /B/Y
```

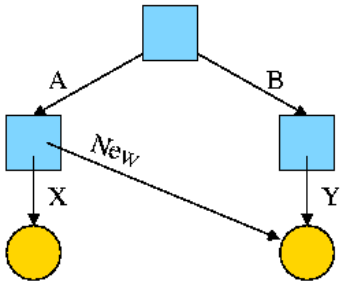


The diagrams in this section use the following conventions

- Yellow circles represent ordinary files.
- Blue squares represent directories.
- Names are written on the edges. For example, one name for the left circle is /A/X.
  - It is not customary to write the names on the edges, normally they are written in the circles and squares.
  - When there are no multi-named files, it doesn't matter if they are written in the node or on the edge.
  - We will see that when files can have multiple names it is much better to write the name on the edge.

Now we execute

```
ln /B/Y /A/New
```



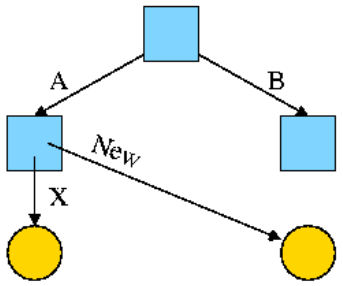
which leads to the next diagram on the right.

At this point there are two equally valid name for the right hand yellow file, /B/Y and /A/New. The fact that /B/Y was created first is **NOT** detectable.

- The directory entries for both file names point to the **same** i-node. **Prove it**
- The file has only one owner (the one who created the file initially). **Prove it**
- The file has one date of last access (the last access by any of its names), one set of permissions, one ... . **Prove it**
- Note the usage: "the file nameS" (plural) vs "the file" (singular).

Assume Bob created /B and /B/Y and Alice created /A, /A/X, and /A/New. Later Bob tires of /B/Y and removes it by executing

```
rm /B/Y
```



The file /A/New is still fine (see third diagram on the right). But it is owned by Bob, who can't find it! If the system enforces quotas Bob will likely be charged (as the owner), but he can neither find nor delete the file (since Bob cannot unlink, i.e. remove, files from /A).

If, prior to removing /B/Y, Bob had examined its "link count" (an attribute of the file), he would have noticed that there is another (hard) link to the file, but would not have been able to determine in which directory (/A in this case) the hard link was located or what is the name of the file in that directory (New in this case).

Since hard links are only permitted to files (not directories) the resulting file system is a dag (directed acyclic graph). That is, there are no directed cycles. We will now proceed to give away this useful property by studying symlinks, which can point to directories.

Symlinks (almost similar to Windows "shortcuts")

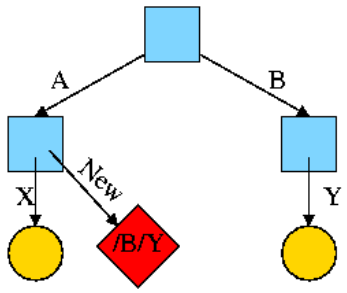
As just noted, hard links do **NOT** create a new file, just another name for an existing file. Once the hard link is created the two names have equal status.

Symlinks, on the other hand **DO** create another file, a non-regular file, that itself serves as another name for the original file. Specifically

- Creation of a symlink results in an *asymmetric* multi-named file. Assuming the original was a regular file, the symlink does indeed have a different status.
- When a symlink is created **another** file is created. The **contents** of the new file is the **name** of the original file.
- A hard link in contrast **is** (another name for) the original file.
- The examples will make this clear.

Again start with an empty file system and this time execute the following code sequence (the only difference from the above is the addition of a -s).

```
cd /
mkdir /A; mkdir /B
touch /A/X; touch /B/Y
ln -s /B/Y /A/New
```



We now have an additional file /A/New, which is a symlink to /B/Y. **The number of files in the system has gone up by 1: Prove it**

- The file named /A/New has the name /B/Y as its *data* (*not* metadata). **Prove it**
- The system notices that /A/New is a red diamond (symlink) so reading /A/New will return the contents of /B/Y (assuming the reader has read permission for /B/Y). It is also possible to read the contents of /A/New itself (those contents are the string "/B/Y"). **Prove both statements**
- The *size* of A/New is 4 bytes, one for each character of its contents. **Prove it**
- If /B/Y is removed, /A/New becomes invalid. **Show this**
- If a new /B/Y is created, A/New is once again valid. **Yes, demonstrate this too**
- Removing /A/New has no effect of /B/Y. **Is this true? why?**
- Examining /B/Y does not reveal the existence of /A/New. **Prove this**
- If a user has write permission for /B/Y, then writing /A/New is possible and writes /B/Y. **Show this**

The bottom line is that, with a hard link, a new **name** is created for the file. This new name has equal status with the original name. This can cause some surprises (e.g., *you* create a link but *I* own the file). With a symbolic link a new **file** is created (owned by the creator naturally) that contains the name of the original file. We often say the new file points to the original file.

Question: Consider the hard link setup above. If Bob removes /B/Y and then creates another /B/Y, what happens to /A/New? **Answer this question**

Question: What about with a symlink? **And this one too**

Note:

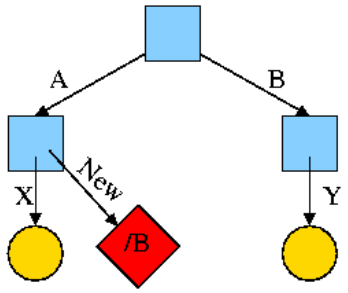
Shortcuts in windows contain more than symlinks contain in unix. In addition to the file name of the original file, they can contain arguments to pass to the file if it is executable. So a shortcut to

```
firefox.exe
```

can specify firefox.exe //google.com

Symlinking a Directory

What happens if the target of the symlink is an existing directory? For example, consider the code below, which gives rise to the diagram on the right.




```
cd /
mkdir /A; mkdir /B
touch /A/X; touch /B/Y
ln -s /B /A/New
```

Answer the three questions below

1. Is there a file named /A/New/Y ?

2. What happens if you execute
   cd /A/New followed by ls

3. What happens if you execute cd /A/New/.. **Use the -L and the -P options for cd and explain your observations**

Please justify all your answers with appropriate shell commands, screen shots and other relevant proof of work!