# Exercise 9 - Solutions

1 (a): If linktofile is a

HARD link: The data is still there because the reference count is >0. It can be accessed via "linktofile" which acts just like any other regular file.

SOFT link: The data is gone (unless there is another hard link to it). The soft link "linktofile" is now dangling as it is now referring to a non-existing file name.

(b) If linktofile is a

HARD link: Just like (a), but now the data is accessible via "fileA"

SOFT link: The link is now gone, but the data is still accessible via "fileA"

(c) HARD link: Need to look at the reference count, if ≥2, there is another link referring to the same data

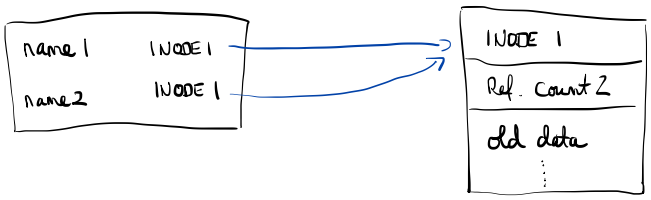
SOFT link: No (other than crawling the entire file system to search for a soft link to the given file, but this is unreliable as mounts may come and go at arbitrary times!)

(d) If the backup system does not check for hard links, it will backup the same data twice, same when restoring → waste of storage space.

For soft links, it is even more tricky if the physical data resides outside the subtree (or partition) that is being backed up. Should the data be written to backup (losing the link structure) or the link (potentially losing the data)?

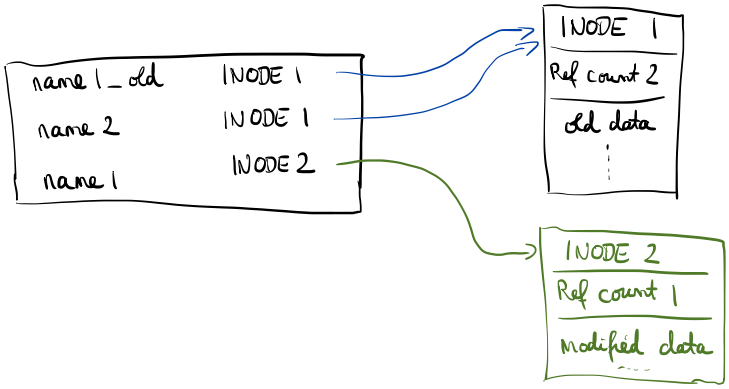
This is a user decision that requires an understanding of the case-specific purpose of the backup

(e) HARD link:

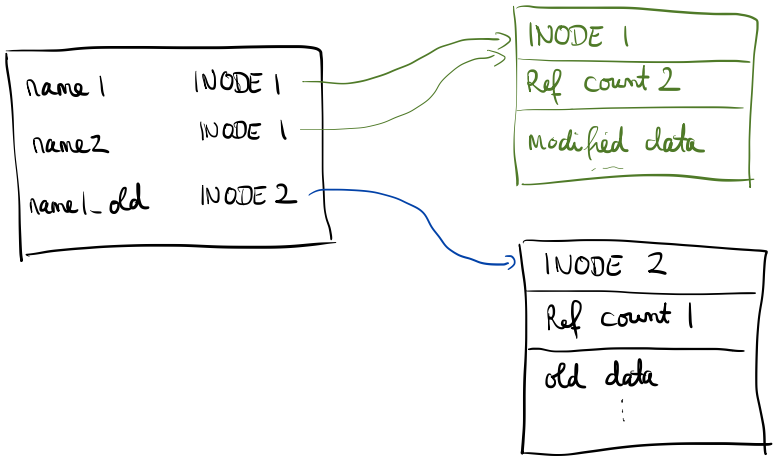



Suppose "name 1" will be edited.

Case 1:
- rename name 1 → name 1_old
- copy name 1_old → name 1
- modify name 1




Case 2:
- copy name 1 → name 1_old
- modify name 1




The result is different (name 2 points to old vs. new data!)

Again, it is a user decision which behavior is right which requires an understanding of the difference!

For SOFT links, this issue is simpler as the association between link and data is path-based, not INODE-based, thus more transparent for the user.

(However, there is the question what it means to copy a symbolic link: should it copy the link or the data? Unix cp opts for copying the data.)

(f) HARD links to directories are often forbidden, SOFT links are allowed. In both cases there is the danger of creating cycles which could cause an infinite loop when trying to traverse a file system. Should be used with care!

(g) Hard link access is as fast as any other file access, soft link access is slower as it requires resolving a second path/file name.

2(a) Total failure happens if two or more disks fail within the rebuild time.

$$\Rightarrow P(\text{total failure}) = 1 - P(0 \text{ or } 1 \text{ disks fail within rebuild time})$$

The number of disks that fail during rebuild time is binomially distributed, i.e.

$$P(\text{total failure}) = 1 - \left[ \binom{n}{0} p^0 (1-p)^n + \binom{n}{1} p^1 (1-p)^{n-1} \right]$$

$$= 1 - \left[ (1-p)^n + np(1-p)^{n-1} \right]$$

(b) If $p$ is small, then $(1-p)^n \approx 1 - np$ and $(1-p)^{n-1} \approx 1 - (n-1)p$

$$\Rightarrow P(\text{total failure}) \approx 1 - \left[ 1 - np + np(1-(n-1)p) \right]$$

$$= n(n-1)p^2$$

If $p$ is small, $n(n-1)p^2 \ll p$ $\Rightarrow$ Huge improvement in reliability

(c) For recovery, we need to read $(n-1) \cdot c \cdot 8$ bits (assuming c is given in bytes), so

$$P(\text{fully recoverable}) = (1 - URE)^{(n-1) \cdot c \cdot 8}$$

$$\approx 1 - (n-1) \cdot c \cdot 8 \cdot URE$$

(d) Here $P(\text{fully recoverable}) \approx 1 - 4 \cdot 2 \cdot 2^{40} \cdot 8 \cdot 10^{-14}$

$$\approx 0.3$$

This is not at all safe!

(e) (i) In this case, data and parity can be computed from the data stream. Therefore:

Need to write $S \cdot (1 + \frac{1}{n-1}) = S \cdot \frac{n-1+1}{n-1} = S \frac{n}{n-1}$ bytes onto n disks in parallel, so speed is $n-1$ the speed of a single disk.

(ii) To compute parity, need to first read blocks already on disk. Note that the new parity can be computed from new block, old block, and old parity via

$$P_{new} = D_{old} \text{ XOR } D_{new} \text{ XOR } P_{old}$$

So need to read old data block $D_{old}$ and old parity $P_{old}$ from disk before writing $P_{new}$ and $D_{new}$.

$\Rightarrow$ At most half the speed of writing to a single disk.