IEEE Standards Interpretation for IEEE Std 1003.1™-1990 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)

Interpretation Request #1

Topic: rename() behavior **Relevant Sections:** 5.5.3.3

Ambiguity in 5.5.3.3 - rename() The words "if either exists" (page 100 line 589) appears to exclude the case where the old and new file did not exist prior to the function call. Is it not the case that if the function call fails the implementation is always required to ensure that both the old and new file states are identical to prior to the call and neither is either created or modified?

A further consideration were implementations that allow rename() to be used across file systems by copying rather than linking, and where cleanup and atomicity is critical.

Interpretation Response

If a call to rename(old, new) returns -1, then the implementation shall in all cases ensure that neither old nor new is created or modified. In particular, if neither old nor new exists prior to the call to rename(), then neither old nor new shall be created by the call. Implementations that support rename() across file systems are bound by the same semantic requirements for such a call to rename() as for a call to rename() within a file system.

Rationale for Interpretation

As is pointed out in the interpretation request, the standard is quite clear and unambiguous in the case where either old or new (or both) exist prior to the call. The only case at issue is when neither exists. The language in Section 5.5.3.3 (which is new in the 1990 revision of the standard) states:

If -1 is returned, neither the file named by old nor the file named by new, if either exists, shall be changed by this function call.

This does not explicitly state what must occur when neither old nor new exists. The interpretation is based on Section 5.5.3.2 (Description), which states (in part):

The rename() function changes the name of the file. The old argument points to the pathname of the file to be renamed. The new argument points to the new pathname of the file.

The rename() function is also specified in the C Standard (X3.159-1989), which in Section 4.9.4.2 states (in part):

The rename function causes the file whose name is the string pointed to by old to be henceforth known by the name given by the string pointed to by new. The file named old is no longer accessible by that name.

Thus, rename() changes file names, but does not change files. Note that in the descriptions of other functions that resolve pathnames but do not create file system objects, the semantics do not explicitly state that the named file must not be created. Yet to create such a file would be considered a semantic error. Examples include unlink(), stat(), chown() and pathconf(). On the other hand, those interfaces that are explicitly designed to create file system objects (such as open(), mkdir() and mkfifo()) document that if -1 is returned, nothing is created. Given the description of the rename() function in 9945-1 and X3.159, it falls into the same category as unlink(), stat() etc. Since file creation is not part of the semantic requirements of rename(), there is no need to document the implicit requirement that a call that fails must not create any extraneous files.