IEEE Standards Interpretation for IEEE Std 1003.1™-2001 IEEE Standard Standard for Information Technology -- Portable Operating System Interface (POSIX®)

Interpretation Request #77
Topic: semaphore names, message queue names **Relevant Sections:** XSH sem_open, sem_unlink, mq_open, mq_unlink,shm_open Page: 1272-1273 Line: 39878-39907 Page: 814 Line: 26691-26693 Page: 1347-1348 Line: 41993-42052

There are two problems:

1. The sem_open() page has some problems regarding how semaphore names relate to the rules for pathnames.

The main problem concerns the values of NAME_MAX and PATH_MAX for semaphore names. The standard states "The name argument conforms to the construction rules for a pathname", and these rules include (via the definitions of "pathname" and "filename") length requirements related to NAME_MAX and PATH_MAX. This is also apparent from the description of the ENAMETOOLONG error on the sem_open() page:

"[ENAMETOOLONG] The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}."

If NAME_MAX or PATH_MAX is not defined as a constant in , how is an application supposed to determine the value that applies to semaphore names? The standard says on the page that in this case the values can be obtained from pathconf(), but semaphore names do not have to appear in the file system, so it is unclear how pathconf() can be used (portably) to do this.

Even if semaphore names do appear in the file system, there is nothing to suggest that the place they appear is the same one that would result from using the semaphore name

as a normal pathname. (I imagine that all implementations that do make semaphores visible in the file system would put them in a different location, otherwise there would be an unacceptable risk of name clashes with normal files in the root directory.)

This means that the value obtained from pathconf(dirname(semaphore_name), _PC_NAME_MAX) might not be the correct value of NAME_MAX. E.g. if semaphores appear somewhere on the /var file system and /var has a different NAME_MAX value from the root file system, then the correct NAME_MAX value could be obtained using pathconf("/var", _PC_NAME_MAX) but there is no portable way for an application to know that it needs to use "/var" as the directory name to pass to pathconf() rather than "/". It also means that the value of PATH_MAX, whether obtained from pathconf() or from , may be too large. E.g. if the implementation prefixes "/var/semaphores" to supplied semaphore names and then just uses the resulting string as a pathname then an ENAMETOOLONG error will occur for semaphore names longer than PATH_MAX minus strlen("/var/semaphores"). Requiring implementations to go through hoops so that semaphore names up to PATH_MAX bytes in length are usable does not seem reasonable.

(There is also a knock-on effect here which creates a problem for the POSIX.13 PSE51 profile of POSIX.1-2001, since PSE51 mandates sem_open() but does not mandate pathconf().)

I considered various solutions to the pathconf() issue, initially based on the idea that the "real" pathname rules should apply if semaphore names appear in the file system, but other rules would apply if they do not. I eventually concluded that there is no advantage for applications in these kind of solutions. Portable applications cannot rely on the use of semaphore names longer than _POSIX_PATH_MAX (or _XOPEN_PATH_MAX on XSI systems), or that have pathname components longer than _POSIX_NAME_MAX (or _XOPEN_NAME_MAX on XSI systems), so the simplest solution is just to make ENAMETOOLONG a "may fail" for exceeding these minimum limits. I believe this would make the standard reflect what happens in current implementations and applications in practice.

For application writers that wish to take advantage of longer semaphore names on systems that support them, and don't mind their applications being non-portable as a result, we can make the actual limits be implementation-defined (thus requiring them to be documented).

An alternative solution would be to introduce new _SC_* constants so that the appropriate values for semaphore names can be obtained from sysconf(), for example _SC_SEM_NAME_MAX and _SC_SEM_PATH_MAX. On a system that places semaphores in the file system under "/var/semaphores", sysconf() could internally do the equivalent of a call to pathconf("/var/semaphores", _PC_NAME_MAX) to obtain the value. On systems where slash characters other than the leading slash are not treated as separating pathname components, sysconf(_SC_SEM_NAME_MAX) would return -1 without setting errno.

Finally, there is another problem that is not a technical issue, but relates to the wording on the sem_open() page. The statement "The interpretation of slash characters other than the leading slash character in name is implementation-defined" is clearly intended as an exception to the construction rules for pathnames when they are applied to semaphore names, since the meaning of non-leading slash characters is defined by the construction rules for pathnames (see XBD6 section 3.266: "It has an optional beginning slash, followed by zero or more filenames separated by slashes."). The text needs to be rearranged to join these two statements with "except that" to make this clear.

Also, the requirement to return ENAMETOOLONG when a pathname component is longer than {NAME_MAX} conflicts with the statement about interpretation of slash characters other than the leading slash.

Note that all of these issues also affect message queue names. Once a decision has been made on how to correct the sem_open() page, equivalent corrections should be made for mq_open().

2. The same issue also affects mq_unlink() and sem_unlink().

Just making the same change to the ENAMETOOLONG error for these functions as was done for mq_open() and sem_open() would solve the problem, but it would also introduce another: it would permit `Weirdnix' implementations where a message queue or semaphore name of a given length can be created, but attempting to unlink it produces an ENAMETOOLONG error. I have attempted to address this by adding an extra requirement on the end of the ENAMETOOLONG descriptions. The wording could probably be improved, and it may be better to add it in a different place.

In sem_open() DESCRIPTION, page 1272 On line 39878 change:

"The name argument conforms to the construction rules for a pathname." to: "The name argument conforms to the construction rules for a pathname, except that the interpretation of slash characters other than the leading slash character in name is implementation-defined, and that the length limits for the name argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}."

On lines 39881-39882 delete: "The interpretation of slash characters other than the leading slash character in name is implementation-defined."

In the ERRORS section Delete lines 39905-39907:

"[ENAMETOOLONG] The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}."

After line 39910 add: "If any of the following conditions occur, the sem_open( ) function may return SEM_FAILED and set errno to the corresponding value: [ENAMETOOLONG] The length of the name argument exceeds _POSIX_PATH_MAX on systems that do not support the XSI option [XSI]or exceeds _XOPEN_PATH_MAX on XSI systems[/XSI], or has a pathname component that is longer than _POSIX_NAME_MAX on systems that do not support the XSI option [XSI]or longer than _XOPEN_NAME_MAX on XSI systems[/XSI]."

If these proposed changes are accepted, also make equivalent changes to the mq_open() page.

2. For mq_unlink Delete lines 26691-26693: "[ENAMETOOLONG] The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}."

After line 26694 add: "The mq_unlink() function may fail if: [ENAMETOOLONG] The length of the name argument exceeds _POSIX_PATH_MAX on systems that do not support the XSI option [XSI]or exceeds _XOPEN_PATH_MAX on XSI systems[/XSI], or has a pathname component that is longer than _POSIX_NAME_MAX on systems that do not support the XSI option [XSI]or longer than _XOPEN_NAME_MAX on XSI systems[/XSI]. A call to mq_unlink() with a name argument that contains the same message queue name as was previously used in a successful mq_open() call shall not give an ENAMETOOLONG error."

On page 1281 line 40127-40129 section sem_unlink: Delete lines 40127-40129: "[ENAMETOOLONG] The length of the name argument exceeds {PATH_MAX} or a pathname component is longer than {NAME_MAX}."

After line 40130 add: "The sem_unlink() function may fail if: [ENAMETOOLONG] The length of the name argument exceeds _POSIX_PATH_MAX on systems that do not support the XSI option [XSI]or exceeds _XOPEN_PATH_MAX on XSI systems[/XSI], or has a pathname component that is longer than _POSIX_NAME_MAX on systems that do not support the XSI option [XSI]or longer than _XOPEN_NAME_MAX on XSI systems[/XSI]. A call to sem_unlink() with a name argument that contains the same semaphore name as was previously used in a successful sem_open() call shall not give an ENAMETOOLONG error."

Interpretation Response #77
The standards states the requirements for semaphore and message queue names, and conforming implementations must conform to this. However, concerns have been raised about this which are being referred to the sponsor.

Rationale for Interpretation
None.