From: Frank Farance
Date: 1995-03-24
Document Number: WG14/N411 X3J11/95-012
Subject: X3J11 Comments on IEEE 1003.1i/D2.0

Review of: ''IEEE Project 1003.1i/D2.0, Standard for Information Technology -- Portable Operating System Interface (POSIX) -- Part 1: System Application Program Interface (API) -- Amendment: Technical Corrigenda to Realtime Extensions [C Langauge]''

Please respond to me via E-mail if you have any further questions.

Thank you for asking us to review your document. The following are suggestions from various technical experts in ANSI X3J11. Considering that the review period was very short, X3J11 reserves the right to provide additional comments on this document in the future.

X3J11 has reviewed a similar document: ''Draft Amendment ISO/IEC 9944-1:1990/DAM1, Information technology -- Portable Operating System Interface (POSIX) -- Part 1: System Application Program Interface (API) [C Language], AMENDMENT 1: Realtime Extension (C Language)''. The following is a copy of our 1994-09-12 comments. Currently, we have no other comments on D2.0.

Page ix, line 93: Typo -- too many quotation marks

Page 11, table: The "Key" field is not defined. It is in 1003.1. You should include the definitions of "(1)" and "(2)", i.e., the explanatory text, in this document to make it easier to review. Otherwise, all other columns in the table are understood.

Page 12, section 2.7.3: You should mention here or somewhere else that the names of the functions are not strictly conforming to ISO/ANSI C because they are not all unique in the first 6 characters. Not all linkers are modern linkers. Even previous UNIX systems would have a problem with "sched_setparam()" and "sched_setscheduler()" because they are not unique in the first 7 characters (assuming C compilers prepend names with "_"). This issue was not a problem in 1003.1 because all the names are unique in the first *6* characters. This problem can be easily overcome by creating macro names (monocase and unique in the first 6 characters) that replace the convenient long names you've defined. You need to let the readers of your document know that you're aware of this limitation in *some* linkers, but your API still can conform to ISO/ANSI C as long as the implementor includes the mapping in the header file.

Page 19, line 66: You should reference "sem_unlink()" since a semaphore may be unlinked upon process termination.

Page 19, line 76: You should reference "mq_unlink()" since a message queue may be unlinked upon process termination.

Page 19, line 76: You make no reference to the shared memory services. What happens to the objects accessed via "shm_open()"?. Part of the problem is that there is no "shm_close()" call. You should mention "shm_unlink()".

Page 29, section 3.4.2.1: Your new model and rationale for extensions to signals misses an important aspect: most library functions (especially C library functions) are not required to be reentrant. Even "sprintf()", "sscanf()", and "memcpy()" can't be used. In real-time applications, you might not be using "fread()" and "fwrite()", but you might want to format error messages (with "sprintf()") or copy blocks of memory (with "memcpy()"). You should require most of the C library functions to be reentrant for your extension. There are some C library functions that are explicitly non-reentrant, e.g., "strtok()". As a rule of thumb, you should require C library functions to be reentrant that meet the following criteria:

- No I/O is performed.
- No signals would be generated with normal parameters (e.g., mathematical functions that cause signals to be generated on overflows, divide by zero, etc.).
- The function is not explicitly non-reentrant.

(This list doesn't *exclude* I/O functions, it just *includes* functions that don't do I/O.)

Obviously, the biggest problem is that you can't use functions like "memcpy()" in signal handlers. Also, it's clear you *intend* to use standard C I/O functions (see page 85, section 8.2.2).

The POSIX project on multi-threading should have a list of functions like this since they have the same problem in multi-threaded applications. You should require the same list (possibly *including* I/O functions) in the list above.

Page 140, section 12.4.2.5.3: The section explains that traditional practice was to use an "mmap()" followed by a "close()". This was true *only* for files that were opened with "open()". However, for System V shared memory, the practice was "shmget()", "shmat()", "shmdt()", "shmctl(IPC_RMID)". You still need a "shm_close()" operation. This completes both paradigms:

File-based:
open
mmap

```
                munmap -- optional
                close

        Memory-based:
                shm_open (like shmget)
                mmap (like shmat)
                munmap (like shmdt) -- optional
                shm_close (like shmctl)
```

Just as you wouldn't expect to close a semaphore with "close", you shouldn't expect to close shared memory with "close". Conversely, you can just overload all of "open" and "close" to include semaphores, message queues, and shared memory. Maybe you only need to overload only close (in this case you'd remove "sem_close()" and "mq_close()") just like the way sockets work. But since most code is specialized in its attachment (open) and detachment (close) phases, i.e., the code has knowledge of what type of object it is accessing and the methods needed to access it (e.g., "open", "shm_open()", "socket()", "application_open()", etc.), the "open-mmap-close" paradigm would only apply to file objects and "shm_open-mmap-shm_close" paradigm would apply to shared memory objects.