The Austin Group Liaison Statement

Date: 2021-04-14
Author: Nick Stoughton

Background

As you are aware, The Austin Group is a collaboration between ISO/IEC JTC 1/SC 22, the Institute for Electical and Electonic Engineers (IEEE), and The Open Group, formed to deal with the ongoing maintenance and development of ISO 9945 POSIX.
The POSIX standard is heavily reliant on the C standard and includes all of the C library interfaces from section 7 of ISO 9899. For each interface that is included in POSIX where it is derived from the C standard, the interface contains the following text:

> The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The Austin Group handles all of the development work for POSIX, including

1. Maintenance. Handling requests for clarification, dealing with areas where the standard differs from existing implementations, etc.
2. Development. From time to time new features are requested, or the handling of a maintenance request results in normative changes. For new material, the Austin Group will typically wordsmith the entire new proposal as a standalone document, and then one of the three member organizations (typically The Open Group) will adopt this document via their own rules (which always includes a ballot of its membership) before it is added to a draft of a revision.

The Austin Group is currently in the midst of a new revision, and several new documents are being added as they are approved. The final text should be ready for ballot by all three member organizations by mid 2023. This revision is known within the Austin Group as “Issue 8”.

Issue 8 - Update to C17

The current version of the POSIX Standard (2018) is based on C99. One of the documents being prepared for inclusion updates the base C standard to C17, with all of the changes from both C11 and C17 brought into the POSIX document. The current proposal is attached to the POSIX Mantis Bug Tracker at https://austingroupbugs.net/view.php?id=1302. The Austin Group invites review and comment from WG 14 on this.

Maintenance Questions

Several outstanding clarification requests deal with text that is directly inherited from C99 (and remains unchanged in C17). Rather than unilaterally attempting to address these from the POSIX point of view, The Austin Group is seeking input from WG 14.

Bug 700: Clarify strtoul's behaviour on strings representing negative numbers

See https://austingroupbugs.net/view.php?id=700.

In which cases does `strtoul` (and other `strtou*` functions) fail with `ERANGE`? How is the range check performed?

This ticket follows the discussion about `strtoul` on austin-group-l in May 2013.

The RETURN VALUE section in POSIX says:

> If the correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX} shall be returned and errno set to [ERANGE].

This text is derived from the wording in the C standard (where the `strtol` , `strtoll` , `strtoul` , and `strtoull` functions are described in one section together), which says (C99 7.20.1.4 p8)

> The `strtol`, `strtoll`, `strtoul`, and `strtoull` functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MIN, LONG_MAX, LLONG_MIN, LLONG_MAX, ULONG_MAX,` or `ULLONG_MAX` is returned (according to the return type and sign of the value, if any), and the value of the macro `ERANGE` is stored in `errno`.

Some people understand this as "if the value I read (the correct value) is outside the `[0, ULONG_MAX]` range, then `strtoul` fails with `ERANGE`".

However, it seems that many (most/all?) implementations do the following:
1. Read independently the optional sign and the subject [0-9]+ sequence
2. If the subject sequence fits in the unsigned long type, then store it in an unsigned long variable. Otherwise (outside the [0, ULONG_MAX] range), fail with ERANGE.
3. If there was a minus sign, then apply negation on the unsigned long variable as if it were a signed long.

According to the discussion on the austin-group-l mailing list, it looks like this is what the POSIX standard (and the C standard) intend to specify. But it looks like it is not clear enough in the specification.

In other words, for 32-bit longs, what should be the value for `x` and `errno` after executing the following code:

```
unsigned long x;
errno = 0;
x = strtoul("-1", NULL, 10);
```

All implementations tested so far set `x` to `0xffffffff` and `errno` remains as 0. Is an implementation permitted to return 0 and set `errno` to `ERANGE`?

Bug 713: in remquo quo should be unspecified when the result is NaN

See https://austingroupbugs.net/view.php?id=713

The description of `quo` in `remquo(x,y,quo)` in POSIX is

> In the object pointed to by `quo`, they store a value whose sign is the sign of `x/y` and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of x/y, where n is an implementation-defined integer greater than or equal to 3. If `y` is zero, the value stored in the object pointed to by `quo` is unspecified.

In the IEC 60559 case `quo` does not seem to be correctly specified when `x` or `y` is ±∞ or `NaN`, only the `y==0` special case is handled.

Since the standard is silent on the value stored in quo in these cases, it is implicitly unspecified. However, to avoid confusion, it is planned to make this explicitly unspecified. Does WG 14 agree, and should the C standard also state this?