|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 7.14.1 | 5 | TE | AG Reference Bug 728 (https://austingroupbugs.net/view.php?id=728)  =======  Title: Restrictions on signal handlers are excessive  The description of the signal function includes the following:  If the signal occurs other than as the result of calling the abort or raise function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object other than by assigning a value to an object declared as volatile sig\_atomic\_t  This is overly restrictive in several cases:   * It does not allow read access to const-qualified objects * It does not allow read access to string literals * It does not allow referencing a modifiable object with static or thread storage duration (that is not a lock-free atomic object) whose last modification was sequenced before the call to the signal handler and whose next modification will be sequenced after the call to the signal handler. | Change the text to:  If the signal occurs other than as the result of calling the abort or raise function, the behavior is undefined if the signal handler refers to any object with static or thread storage duration that is not a lock-free atomic object, not a const-qualified object, and not a string literal, other than by assigning a value to an object declared as volatile sig\_atomic\_t, unless the previous modification (if any) to the object happens before the signal handler is called and the return from the signal handler happens before the next modification (if any) to the object |  |
|  |  | 7.23.2 | 7-8 | TE | AG Reference Bug 689 (https://austingroupbugs.net/view.php?id=689)  =======  Title: Possibly unintended allowance for stdio deadlock  7.23.2 Streams states:  7 Each stream has an associated lock that is used to prevent data races when multiple threads of execution access a stream, and to restrict the interleaving of stream operations performed by multiple threads. Only one thread may hold this lock at a time. The lock is reentrant: a single thread may hold the lock multiple times at a given time.  8 All functions that read, write, position, or query the position of a stream lock the stream before accessing it. They release the lock associated with the stream when the access is complete.  and 7.23.3 Files states in para 3:  When a stream is line buffered, characters are intended to be transmitted to or from the host environment as a block when a new-line character is encountered. Furthermore, characters are intended to be transmitted as a block to the host environment when a buffer is filled, when input is requested on an unbuffered stream, or when input is requested on a line buffered stream that requires the transmission of characters from the host environment.  Although support for the latter is implementation-defined, if the "when input is requested" parts are implemented, it creates the potential for deadlock.  For example, if thread A is holding the lock associated with a line-buffered output stream and its progress is blocked waiting for thread B to do something, and thread B happens to use stdio for reading any unbuffered (or line buffered with an empty buffer) stream as part of its operation, the requirement in 7.23.2 para 8 means the program will deadlock. This behavior seems highly undesirable and unintended. | Requiring deadlock detection seems too onerous, given that POSIX makes it optional for pthread\_mutex\_lock, but perhaps there ought at least to be an attempt at detection. The question is then what to do if deadlock is not detected but nor has it been established that a deadlock situation does not exist. Since implementing support for the flush is optional anyway, just not doing the flush seems like an acceptable solution.  After:  All functions that read, write, position, or query the position of a stream lock the stream before accessing it. They release the lock associated with the stream when the access is complete.  add:  If the lock is not immediately available, the function waits for it to become available, except in the following circumstances. If the stream is line buffered and is open for writing or for update, and the reason the function is attempting to lock the stream is because it is going to request input on another stream that is unbuffered, or is line buffered and requires the transmission of characters from the host environment (see 7.23.3), then the function attempts to determine whether a deadlock situation exists. If a deadlock situation is found to exist, the function shall fail. If the function is able to establish that a deadlock situation does not exist, it shall wait for the lock to become available. If the function does not establish whether or not a deadlock situation exists, it shall continue as if it had already locked the stream, found its buffer to be empty, and released the lock. |  |
|  |  | 7.24.1.7 | 5 | TE | AG Reference Bug 700 (https://austingroupbugs.net/view.php?id=700)  =======  Title: strtol cannot return LONG\_MIN with two's complement long  The description of strtol, strtoll, strtoul, and strtoull states:  If the subject sequence begins with a minus sign, the value resulting from the conversion is negated (in the return type).  The parenthetical phrase "(in the return type)" was added in C99 in response to DR #006 http://www.open-std.org/jtc1/sc22/wg14/docs/rr/dr\_006.html  This clarified the behavior of strtoul but it broke strtol, because with two's complement signed long, it is not possible to produce the value LONG\_MIN by negating a positive value "in the return type".  Likewise for the equivalent wcsto\* functions. | Change the text to:  If the subject sequence begins with a minus sign, the value resulting from the conversion is negated; for functions whose return type is an unsigned integer type this negation is performed in the return type.  Make the same change for wcstol, wcstoll, wcstoul, and wcstoull. |  |
|  |  | 7.24.7 | 1 | TE | AG Reference Bug 708 (https://austingroupbugs.net/view.php?id=708)  =======  Title: mblen, mbtowc, and wctomb data races  As per https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2396.htm#dr\_498 it seems that in Oct 2018 the committee agreed in principle with the goal of N2281, and solicited a new paper from the author.  Hopefully a new paper has been submitted and will lead to a satisfactory outcome. However, if that is not the case then the Austin Group strongly recommends that the fallback position should not be the status quo, but to align with POSIX.  The current situation is that the C standard requires mblen, mbtowc, and wctomb to avoid data races, but POSIX says that they need not be thread-safe. POSIX currently still refers to C99, so this will not become a problem until the next POSIX revision, which will refer to C17, is approved. (It is in the late stages of development.)  Since DRs for C17 are not being accepted, in order not to force POSIX implementations to change, the next POSIX revision will need to state that it does not defer to C17 regarding thread-safety of these functions. Hopefully the revision after next will be able to return to deferring to the C standard, but this will depend on whether, and under what conditions, the relevant future C standard still requires them to avoid data races.  If wording cannot be agreed for C23 that specifies the precise conditions under which these functions avoid data races, then the standard should simply change to match POSIX and state that they need not avoid data races. | Append to 7.24.7 para 1:  These functions are not required to avoid data races with other calls to the same function. |  |
|  |  | 7.29.3.5 | 3 | TE | Section 7.29.3.5 Para 3  AG Reference Bug 739 (https://austingroupbugs.net/view.php?id=739)  =======  Title: strftime %F conversion claims to provide ISO 8601 date format but does so only for a limited year range  The strftime %F conversion is described as:  %F is equivalent to "%Y-%m-%d" (the ISO 8601 date format)  However, if the year is between 0 and 999 this produces at most a three digit year, whereas ISO 8601 specifies a minimum of four digits for years in that range.  Also, if the year is outside the range 0 to 9999, according to wikipedia "To represent years before 0000 or after 9999, [ISO 8601] also permits the expansion of the year representation but only by prior agreement between the sender and the receiver. An expanded year representation [±YYYYY] must have an agreed-upon number of extra year digits beyond the four-digit minimum, and it must be prefixed with a + or − sign". | There are three different solutions, depending on how much equivalence to ISO 8601 is to be claimed.  Suggested resolution:  Option 1 - only claim ISO 8601 equivalence for years 1000 to 9999  Change the %F description to:  %F is equivalent to "%Y-%m-%d" (the ISO 8601 date format, when the year is between 1000 and 9999 inclusive)  Option 2 - only claim ISO 8601 equivalence for years 0 to 9999  Change the %F description to:  %F is equivalent to "%Y-%m-%d", except that the stored year is filled  as needed with leading zeros so that if the year is between 0 and 999 inclusive, four digits are stored. (This provides the ISO 8601 date format when the year is between 0 and 9999 inclusive.)  Option 3 - full ISO 8601 equivalence  Since "An expanded year representation [±YYYYY] must have an agreed-upon number of extra year digits beyond the four-digit minimum", there needs to be a way for that agreed-upon number to be used in the strftime format string. This would require adding field widths: the wording could be adapted from POSIX.1-2017. The requirement that the year must be prefixed with a + or − sign could be handled either by adding the + flag from POSIX.1-2017 or by stating the need for a + sign for years > 9999 in the description of %F. |  |
|  |  | 7.29.2.3 | 3 | TE | AG Reference Bug 1614 (https://austingroupbugs.net/view.php?id=1614)  ========  Title: meaning of (time\_t)-1 return from mktime  The mktime description states, under "Returns":  The mktime function returns the specified calendar time encoded as a value of type time\_t. If the calendar time cannot be represented, the function returns the value (time\_t)(-1).  An application writer reading this is likely to infer from the way it is worded that when mktime returns (time\_t)-1 it means that the calendar time to be returned was not representable.  Indeed, searching open source applications for calls to mktime turns up many uses where a return of (time\_t)-1 is assumed to indicate this. For example, the Python time module turns an error return of (time\_t)-1 from the C library mktime function into a PyExc\_OverflowError exception. A limited search also found no applications that treat a return of (time\_t)-1 as possibly indicating some other kind of failure, although an extensive search might do so.  However, this interpretation of the "Returns" text seems to be at odds with the committee's response in 1994 to DR #136, which says that mktime can return (time\_t)-1 for broken-down times that refer to times in the "spring-forward gap" (see https://www.open-std.org/jtc1/sc22/wg14/www/docs/dr\_136.html)  One way to reconcile the two is to observe that, since local time and Daylight Saving Time are implementation-defined, an implementation could define them in such a way that times in the spring-forward gap are converted to a value that cannot be represented. For example, it could say they are converted to UINT64\_MAX if time\_t is a signed 64-bit integer type. Then the C standard would require mktime to return (time\_t)-1 because UINT64\_MAX can't be represented in that time\_t type.  If this was the committee's reasoning in 1994, then it would be helpful to have this confirmed. Another possibility is that they simply overlooked the significance of the way the "Returns" clause is worded.  Deciding what the standard currently requires is one thing, but there is also the (perhaps more important) matter of what C23 should require.  The current state of applications is the result of a combination of factors:  1. Application writers interpreting the "Returns" text in the way described above.  2. Almost all implementations not returning (time\_t)-1 for times in the spring-forward gap, with the consequence that, over the decades, applications have been mostly developed and run on such systems. This is known because the original NIST-PCTS tested for this behavior (as stated in DR #136) and The Open Group test suite, which has been used to certify dozens of systems as POSIX conforming (since 2003) and (since 1990) as XPG3, XPG4, and UNIX conforming, also tests for it. The list of certified systems includes Microsoft Windows (NT 3.5, 3.51 and 4.0 were certified POSIX conforming). Running the example program from DR #136 on some non-certified systems identified that glibc and FreeBSD do not return (time\_t)-1. So far NetBSD is the only system that has been confirmed as returning (time\_t)-1, but it can actually behave both ways: there is a NO\_ERROR\_IN\_DST\_GAP compile time option to control it (with the (time\_t)-1 return as the default). DR #136 says that Arthur David Olson's popular "tz" time zone software returned (time\_t)-1 (in 1994) and this may have influenced the response to that DR, as it is known to have been adopted by many systems. However, it is now evident that almost all (if not all) systems that adopted it modified it so that it does not return (time\_t)-1.  3. Even when run on an implementation that does return (time\_t)-1 for times in the spring-forward gap, occurrences of this condition are rare, and the occasional application misbehavior (by treating it the same as the "cannot be represented" case) may have gone unnoticed.  Given that almost all implementations of mktime do not return (time\_t)-1 for times in the spring-forward gap, and that applications which can handle a return of (time\_t)-1 appropriately for any condition other than the calendar time to be returned being unrepresentable seem to be exceedingly rare, it would benefit application portability if C23 upholds the meaning of the (time\_t)-1 return as being that the calendar time to be returned cannot be represented and disallows returning (time\_t)-1 for other reasons.  If mktime can return (time\_t)-1 for other reasons, this creates a problem for applications if they want to to distinguish the different cases. If (time\_t)-1 is returned when tm\_isdst is negative, they can try a second call with tm\_isdst set to 0 (or 1) and assume that if this succeeds the original return of (time\_t)-1 was caused by a DST transition, but this additional code is unnecessary on almost all systems, and it only handles that one additional case. If (time\_t)-1 is returned when tm\_isdst is not negative, how are applications to distinguish the "cannot be represented" case from other cases that they might prefer to treat as non-fatal?  If the source of the broken-down time was from a file or database, or user input, then perhaps it is not much of a problem if the return of (time\_t)-1 is treated as a fatal error (with a misleading error message), but when mktime is used to perform manipulations of the struct tm members, it is more of a problem. DR #136 suggests that tm\_isdst is left as 0 or 1 when doing such manipulations, and that may be true when the time adjustment is small, but when adding or subtracting any whole number of days, setting tm\_isdst to -1 is a perfectly reasonable thing for an application to do.  Not returning (time\_t)-1 is simply better for applications. The glibc source contains this comment about it:  The requested time probably falls within a spring-forward gap of size DT. Follow the common practice in this case, which is to return a time that is DT away from the requested time, [...] In practice, this is more useful than returning -1.  On a system which does not return (time\_t)-1, if an application wants to detect whether the broken-down time is in a spring-forward gap, all it needs to do is look for appropriate changes to the struct tm fields after mktime returns. (In the example from DR #136, tm\_hour changes from 2 to either 1 or 3.)  Finally, there is also a subtle problem with the "cannot be represented" wording. It is not clear if it means cannot be represented in a time\_t, or cannot be represented in the time\_t encoding used for the return value. Microsoft Windows, and perhaps some other systems, uses a time\_t encoding that does not include negative values (even though its time\_t is signed), and thus returns (time\_t)-1 if the calculated calendar time is negative. To ensure this is clearly allowed, the wording should be changed to "cannot be represented in the time\_t encoding used for the return value". | Four options are given based on two independent decisions (what C17 requires and what C23 should require). The Austin Group has a strong preference for options 1 and 2 over options 3 and 4.  Option 1  A return of (time\_t)-1 means that the calendar time to be returned could not be represented. Implementations can return (time\_t)-1 for times in the spring-forward gap by defining local time and Daylight Saving Time in such a way that times in the spring-forward gap are converted to a value that cannot be represented, but for the sake of application portability this loophole should be closed in C23 by changing:  ... not restricted to the ranges indicated above. 389) On successful completion, the values of the tm\_wday and tm\_yday components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated above; the final value of tm\_mday is not set until tm\_mon and tm\_year are determined.  to:  ... not restricted to the ranges indicated above. If the local time to be used for the conversion is one that includes Daylight Saving Time adjustments, a positive or zero value for tm\_isdst causes the mktime function to perform the conversion as if Daylight Saving Time, respectively, is or is not in effect for the specified time. A negative value causes it to attempt to determine whether Daylight Saving Time is in effect for the specified time; if it determines that Daylight Saving Time is in effect it produces the same result as an equivalent call with a positive tm\_isdst value, otherwise it produces the same result as an equivalent call with a tm\_isdst value of zero. 389) On successful completion, the components of the structure are set to the same values that would be returned by a call to the localtime function with the calculated calendar time as its argument.  and changing footnote 389 to read:  If the broken-down time specifies a time that is either skipped over or repeated when a transition to or from Daylight Saving Time occurs, it is unspecified whether the mktime function produces the same result as an equivalent call with a positive tm\_isdst value or as an equivalent call with a tm\_isdst value of zero.  Also, under "Returns" change:  If the calendar time cannot be represented  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value  Option 2  A return of (time\_t)-1 can mean other things than that the calendar time to be returned could not be represented, because [insert explanation here], but for the sake of application portability C23 should disallow this by changing:  ... not restricted to the ranges indicated above. 389) On successful completion, the values of the tm\_wday and tm\_yday components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated above; the final value of tm\_mday is not set until tm\_mon and tm\_year are determined.  to:  ... not restricted to the ranges indicated above. If the local time to be used for the conversion is one that includes Daylight Saving Time adjustments, a positive or zero value for tm\_isdst causes the mktime function to perform the conversion as if Daylight Saving Time, respectively, is or is not in effect for the specified time. A negative value causes it to attempt to determine whether Daylight Saving Time is in effect for the specified time; if it determines that Daylight Saving Time is in effect it produces the same result as an equivalent call with a positive tm\_isdst value, otherwise it produces the same result as an equivalent call with a tm\_isdst value of zero. 389) On successful completion, the components of the structure are set to the same values that would be returned by a call to the localtime function with the calculated calendar time as its argument.  and changing footnote 389 to read:  If the broken-down time specifies a time that is either skipped over or repeated when a transition to or from Daylight Saving Time occurs, it is unspecified whether the mktime function produces the same result as an equivalent call with a positive tm\_isdst value or as an equivalent call with a tm\_isdst value of zero.  Also, under "Returns" change:  If the calendar time cannot be represented  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value  Option 3  A return of (time\_t)-1 means that the calendar time to be returned could not be represented. Implementations can return (time\_t)-1 for times in the spring-forward gap by defining local time and Daylight Saving Time in such a way that times in the spring-forward gap are converted to a value that cannot be represented, but they should not need to use this loophole to do so, and C23 should change:  If the calendar time cannot be represented, the function returns the value (time\_t)(-1).  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value, or if the function does not succeed for some other reason, the function returns the value (time\_t)(-1).  Option 4  A return of (time\_t)-1 can mean other things than that the calendar time to be returned could not be represented, because [insert explanation here], and this should be clarified in C23 by changing:  If the calendar time cannot be represented, the function returns  the value (time\_t)(-1).  to:  If the calendar time cannot be represented in the time\_t encoding used for the return value, or if the function does not succeed for some other reason, the function returns the value (time\_t)(-1). |  |