N3060: C2y `fopen "p"` and bring `fopen`'s `mode` closer to POSIX

Document #: N3060
Date: 2022-09-26
Project: Programming Language C
Reply-to: Niall Douglas
<s_sourceforge@nedprod.com>

In the May 2022 WG14 meeting we discussed [N2857] *C2x `fopen("x")` and `fopen("a")` v2*, and the committee sought:

1. Fixes to the proposed normative wording changes, specifically:
   (a) Add mention of processes in statements about atomicity of checks.
   (b) Remove mention of atomicity applies to other users of `fopen` only, as other syscalls may be used by other threads or functions.
   (c) 'must' => 'shall'.
2. That the second half of the permitted implementation of the existing C11 `fopen("x")` be split out into a standalone `fopen` letter, for which I have chosen 'p' (private).

I believe that these solve the committee's concerns raised in the meeting about ensuring TOCTOU safety both in terms of file content use, and file naming on the file system.

'Real life' then intruded and I was unable to deliver the next revision of this paper before the C23 IS cutoff, so I have split the paper into two:

- The uncontroversial bits already agreed by the committee as a C23 IS delta suitable for NB comment (NOT this paper).
- The bits perhaps needing another round by the committee targeting post-C23, which include taking the opportunity to reconcile the `fopen` modifier specification with that from the next release of POSIX (this paper).

Contents

1 Proposed improved wording **2**
1.1 7.21.5.3.3 2
1.2 7.21.5.3.5 3
1.3 7.21.5.3.7 (insertion) 3

2 Platform compatibility **4**
2.1 `fopen('p')` 4

3 Acknowledgements 4

4 References 4

1 Proposed improved wording

1.1 7.21.5.3.3

~~The argument mode points to a string. If the string is one of the following, the file is open in the indicated mode. Otherwise, the behavior is undefined.~~

~~r open text file for reading~~
~~w truncate to zero length or create text file for writing~~
~~wx create text file for writing~~
~~a append; open or create text file for writing at end-of-file~~
~~rb open binary file for reading~~
~~wb truncate to zero length or create binary file for writing~~
~~wbx create binary file for writing~~
~~ab append; open or create binary file for writing at end-of-file~~
~~r+ open text file for update (reading and writing)~~
~~w+ truncate to zero length or create text file for update~~
~~w+x create text file for update~~
~~a+ append; open or create text file for update, writing at end-of-file~~
~~r+b or rb+ open binary file for update (reading and writing)~~
~~w+b or wb+ truncate to zero length or create binary file for update~~
~~w+bx or wb+x create binary file for update~~
~~a+b or ab+ append; open or create binary file for update, writing at end-of-file~~

The argument `mode` points to a string. The behavior is unspecified if any character occurs more than once in the string. If the string begins with one of the following characters, then the file shall be opened in the mode indicated. Otherwise the behavior is undefined.

`r` open file for reading
`w` truncate to zero length or create file for writing
`a` append; open or create file for writing at end-of-file

The remainder of the string can contain any of the following characters in any order, and further affect how the file is opened:

`b` The contents of the file shall be treated as binary, otherwise they shall be treated as text.
`x` The file shall be opened in exclusive mode.
`p` The file shall be opened in private mode.
`+` The file shall be opened for update (both reading and writing), rather than just reading or writing.

[*Note:* This is derived largely from the proposed changes to the next POSIX standard, courtesy of Nick Stoughton. – end note]

1.2 7.21.5.3.5

Opening a file with exclusive mode (`'x'` as ~~the last~~ a character in the mode argument) fails if the file already exists or cannot be created. The check for the existence of the file and the creation of the file if it does not exist is atomic with respect to other threads and processes. If the implementation is not capable of performing the check for the existence of the file and the creation of the file atomically, it shall fail instead of performing a non-atomic check and creation.

1.3 7.21.5.3.7 (insertion)

Opening a file with private mode (`'p'` as a character in the mode argument) creates a file whose contents cannot be accessed by other means at any point in time including after program end, and whose `filename` may be ignored by implementations except to identify which storage ought to hold the file. If the implementation is not capable of creating private files, it shall fail.

[*Note:* This would be equivalent to proposed `std::file_handle::temp_inode()` in C++. It effectively creates your own private swap file that nobody else can access, and which vanishes upon close. I know that some on the committee don't see much point in such a facility, however if you ever write code where physical RAM is a constraint (e.g. mobile phones, embedded systems, GPUs), and one needs to manually tell a kernel what large data sets ought to be preferentially kicked out to storage, this facility is a real boon. Another use case is for secrets which you don't want to keep within process memory.

For security reasons, implementations can NOT implement this by creating a file entry visible outside the calling process and then immediately unlinking it unless they have some other implementation specific mechanism of preventing another process from opening the file for access in the very small window between file entry creation and removal.

Microsoft Windows, for example, has the concept of 'share mode' and you can create a new randomly named file entry which nobody else can open, then set the delete-on-close flag to ensure the file's contents are disposed of upon close, and since Windows 10 one can also remove the file entry from the filing system so nobody else can find it thereafter anyway. This would be an acceptable implementation.

On Linux, there is the proprietary extension flag `O_TMPFILE` which creates an unnamed temporary regular file, this would also be acceptable.

If on straight POSIX, if a suid process had temporarily dropped privileges or is running as `root`, a file could be created with no access privileges and owned by `root` to prevent anybody else opening it in the short window before it is unlinked. That would also be an acceptable implementation. – end note]

2 Platform compatibility

The relaxed requirements on `fopen` `mode` character ordering match those of the next edition of POSIX, and to this author's best knowledge have been a common relaxation on many implementations for many years now.

2.1 `fopen('p')`

I checked the following implementations to see if `'p'` is used by any of them:

- Linux (glibc): `'p'` is unused (extensions are `'c'`, `'e'`, `'m'` and `'x'`).
- FreeBSD: `'p'` is unused (extensions are `'e'` and `'x'`).
- NetBSD: `'p'` is unused (extensions are `'e'`, `'f'`, `'l'` and `'x'`).
- OpenBSD: `'p'` is unused (extensions are `'e'` and `'x'`).
- MacOS: `'p'` is unused (extensions are `'x'`).
- Microsoft VS2019: `'p'` is unused (extensions are `'c'`, `'n'`, `'t'` and `'x'`).
- QNX: `'p'` is unused (no extensions).
- HPUX: `'p'` is unused (no extensions).

3 Acknowledgements

Thanks to Robert Seacord for his help in drafting the proposed normative wording. Thanks to Aaron Ballman for reminding me of the existence of [N2357]. Thanks to Nick Stoughton for writing the original paper raising this issue, and to Joseph Myers for his feedback on earlier drafts.

4 References

[N2357] Stoughton, Nick
Change Request for fopen exclusive access
`http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2357.htm`

[N2731] *C2x Working Draft*
`http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2731.pdf`

[N2857] Douglas, Niall
C2x `fopen("x")` and `fopen("a")` v2
`http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2857.pdf`

[POSIX.2017] *The 2017 POSIX standard*
https://pubs.opengroup.org/onlinepubs/9699919799.2018edition/functions/contents.html