N3059: C23 `fopen "x"` and `"a"`

Document #:	N3059
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 (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 (NOT this paper).

Contents

1 Proposed DR wording

1.1 7.21.5.3.5

Opening a file with exclusive mode (`'x'` as the last character in the mode argument) fails if the file already exists or cannot be created. ~~Otherwise, the file is created with exclusive (also known as non-shared) access to the extent that the underlying system supports exclusive access.~~ 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.

> [*Note:* The last sentence is important: if a program is written assuming that the check is atomic, and it is not atomic, then data loss or corruption would occur. It is better to return an error here so the program can adapt rather than silently allow data loss or corruption. – end note]

1.2 7.21.5.3.6

Opening a file with append mode (`'a'` as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file at the point of buffer flush or actual write, regardless of intervening calls to the `fseek` ~~function~~, `fsetpos`, or `rewind` functions. Incrementing the current end-of-file by the amount of data written is atomic with respect to other threads writing to the same file provided the file was also opened in append mode. If the implementation is not capable of incrementing the current end-of-file atomically, it shall fail instead of performing non-atomic end-of-file writes. In some implementations, opening a binary file with append mode (`'b'` as the second or third character in the above list of `mode` argument values) may initially position the file position indicator for the stream beyond the last data written, because of null character padding.

> [*Note:* This text only guarantees the atomicity of the increment of the end of file, NOT the atomicity of the write of the data. This difference is important: no additional locking is needed here on platforms capable of atomic integer increment. – end note]

1.3 K.3.5.2.1.7

~~To the extent that the underlying system supports the concepts, files opened for writing shall be opened with exclusive (also known as non-shared) access. If the file is being created, and the first~~

~~character of the mode string is not 'u' , to the extent that the underlying system supports it, the file shall have a file permission that prevents other users on the system from accessing the file. If the file is being created and first character of the mode string is 'u' , then by the time the file has been closed, it shall have the system default file access permissions.~~

> [*Note:* Robert Seacord suggested that this ought to be removed for consistency with the change above. Me personally I am agnostic, but given that the only implementation that I know of of `fopen_s` which is Microsoft's, it would now conform to C2x if this stanza is removed. – end note]

2 Platform compatibility

I checked whether the proposed new wording would break any existing platforms implementing C11:

2.1 `fopen('x')`

- Linux (glibc): Existing implementation is compatible.
- FreeBSD: Existing implementation is compatible.
- NetBSD: Existing implementation is compatible.
- OpenBSD: Existing implementation is compatible.
- MacOS: Existing implementation is compatible.
- Microsoft VS2019: Existing implementation is compatible.
- QNX: `fopen('x')` not supported. `open()` is compatible.
- HPUX: `fopen('x')` not supported. `open()` is compatible.

The excellent compatibility story here is almost certainly due to POSIX `O_EXCL` creating an easy choice for how to implement `fopen('x')`.

2.2 `fopen('a')`

- glibc implements `fopen('a')` as `O_APPEND`, so appends are atomic across the system as per the proposed wording.
  https://sourceware.org/git/?p=glibc.git;a=blob;f=libio/fileops.c;h=0986059e7b16f885f8ab62bc9hb=HEAD#l237.
- BSD libc implements `fopen('a')` as `O_APPEND`, so appends are atomic across the system as per the proposed wording.
  https://svnweb.freebsd.org/base/head/lib/libc/stdio/flags.c?revision=326025&view=markup#l72
- Microsoft UCRT implements `fopen('a')` as `_O_APPEND`:

```
    case 'a':
        result._lowio_mode = _O_WRONLY | _O_CREAT | _O_APPEND;
        result._stdio_mode = _IOWRITE;
        break;
```

Then:

```
    // Set FAPPEND flag if appropriate. Don't do this for devices or pipes:
    if ((options.crt_flags & (FDEV | FPIPE)) == 0 && (oflag & _O_APPEND))
        _osfile(*pfh) |= FAPPEND;
```

Then:

```
    if (_osfile(fh) & FAPPEND)
        (void)_lseeki64_nolock(fh, 0, FILE_END);
```

Which eventually calls Win32 `SetFilePointerEx()`. This means appends are atomic within the local process per file descriptor, but are not atomic per inode in the local process, nor atomic across the system.

I suspect that this is an implementation oversight considering there are two forms of whole system atomic append supported on Windows:

1. Win32 `CreateFile()` when opened with `GENERIC_READ | FILE_WRITE_ATTRIBUTES | STANDARD_RIGHTS_WRITE | FILE_APPEND_DATA` instead of `GENERIC_READ | GENERIC_WRITE` does perform atomic appends across the system.
2. Win32 `WriteFile()` when supplied with an offset to write value of all bits one will perform an atomic append for that specific write across the system.

Steve Wishnousky from Microsoft who helps maintain their UCRT doesn't see any major impact from ensuring the file access is atomic (stated on the WG21-WG14 liason mailing list, 11th Oct 2021).

The source code of other platform's `fopen()` implementation was not easily available to me, so I cannot say more about how those implement `fopen('a')`.

2.3 `fopen_s('x')`

- Linux (glibc): `fopen_s()` is not provided.
- FreeBSD: `fopen_s()` is not provided.
- NetBSD: `fopen_s()` is not provided.
- OpenBSD: `fopen_s()` is not provided.
- MacOS: `fopen_s()` is not provided.
- Microsoft VS2019: Existing implementation is compatible.
- QNX: `fopen_s()` is not provided.

- HPUX: `fopen_s()` is not provided.

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