Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'getc_unlocked.3p' command

$ man getc_unlocked.3p

GETC_UNLOCKED(3P) POSIX Programmer's Manual GETC_UNLOCKED(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked ? stdio with explicit client locking

SYNOPSIS

```
#include <stdio.h>
int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

DESCRIPTION

Versions of the functions getc(), getchar(), putc(), and putchar() re? spectively named getc_unlocked(), getchar_unlocked(), putc_unlocked(), and putchar_unlocked() shall be provided which are functionally equiva? lent to the original versions, with the exception that they are not re? quired to be implemented in a fully thread-safe manner. They shall be thread-safe when used within a scope protected by flockfile() (or ftry? lockfile()) and funlockfile(). These functions can safely be used in a

multi-threaded program if and only if they are called while the invok?
ing thread owns the (FILE *) object, as is the case after a successful
call to the flockfile() or ftrylockfile() functions.

If getc_unlocked() or putc_unlocked() are implemented as macros they
may evaluate stream more than once, so the stream argument should never
be an expression with side-effects.

RETURN VALUE

See getc(), getchar(), putc(), and putchar().

ERRORS

See getc(), getchar(), putc(), and putchar().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since they may be implemented as macros, getc_unlocked() and putc_un?
locked() may treat incorrectly a stream argument with side-effects. In
particular, getc_unlocked(*f++) and putc_unlocked(c,*f++) do not neces?
sarily work as expected. Therefore, use of these functions in such
situations should be preceded by the following statement as appropri?
ate:

```
#undef getc_unlocked
#undef putc_unlocked
```

RATIONALE

Some I/O functions are typically implemented as macros for performance
reasons (for example, putc() and getc()). For safety, they need to be
synchronized, but it is often too expensive to synchronize on every
character. Nevertheless, it was felt that the safety concerns were more
important; consequently, the getc(), getchar(), putc(), and putchar()
functions are required to be thread-safe. However, unlocked versions
are also provided with names that clearly indicate the unsafe nature of
their operation but can be used to exploit their higher performance.
These unlocked versions can be safely used only within explicitly
locked program regions, using exported locking primitives. In particu?

lar, a sequence such as:

```
flockfile(fileptr);
putc_unlocked('1', fileptr);
putc_unlocked('\n', fileptr);
fprintf(fileptr, "Line 2\n");
funlockfile(fileptr);
```

is permissible, and results in the text sequence:

```
1
Line 2
```

being printed without being interspersed with output from other threads.

It would be wrong to have the standard names such as getc(), putc(), and so on, map to the ``faster, but unsafe'' rather than the ``slower, but safe'' versions. In either case, you would still want to inspect all uses of getc(), putc(), and so on, by hand when converting existing code. Choosing the safe bindings as the default, at least, results in correct code and maintains the ``atomicity at the function'' invariant. To do otherwise would introduce gratuitous synchronization errors into converted code. Other routines that modify the stdio (FILE *) struc? tures or buffers are also safely synchronized.

Note that there is no need for functions of the form getc_locked(), putc_locked(), and so on, since this is the functionality of getc(), putc(), et al. It would be inappropriate to use a feature test macro to switch a macro definition of getc() between getc_locked() and getc_unlocked(), since the ISO C standard requires an actual function to exist, a function whose behavior could not be changed by the feature test macro. Also, providing both the xxx_locked() and xxx_unlocked() forms leads to the confusion of whether the suffix describes the behav? ior of the function or the circumstances under which it should be used.

Three additional routines, flockfile(), ftrylockfile(), and funlock? file() (which may be macros), are provided to allow the user to delin? eate a sequence of I/O statements that are executed synchronously.

The ungetc() function is infrequently called relative to the other

functions/macros so no unlocked variation is needed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Standard I/O Streams, flockfile(), getc(), getchar(), putc(), putchar()

The Base Definitions volume of POSIX.1?2017, <stdio.h>

COPYRIGHT