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 'pthread_attr_destroy.3p' command

$ man pthread_attr_destroy.3p

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

pthread_attr_destroy, pthread_attr_init ? destroy and initialize the thread attributes object

SYNOPSIS

```
#include <pthread.h>
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

DESCRIPTION

The pthread_attr_destroy() function shall destroy a thread attributes object. An implementation may cause pthread_attr_destroy() to set attr to an implementation-defined invalid value. A destroyed attr attributes object can be reinitialized using pthread_attr_init(); the results of otherwise referencing the object after it has been destroyed are unde? fined.

The pthread_attr_init() function shall initialize a thread attributes object attr with the default value for all of the individual attributes used by a given implementation.

The resulting attributes object (possibly modified by setting individ? ual attribute values) when used by pthread_create() defines the at? tributes of the thread created. A single attributes object can be used in multiple simultaneous calls to pthread_create(). Results are unde? fined if pthread_attr_init() is called specifying an already initial? ized attr attributes object.

The behavior is undefined if the value specified by the attr argument to pthread_attr_destroy() does not refer to an initialized thread at? tributes object.

RETURN VALUE

Upon successful completion, pthread_attr_destroy() and pthread_attr_init() shall return a value of 0; otherwise, an error num? ber shall be returned to indicate the error.

ERRORS

The pthread_attr_init() function shall fail if:

ENOMEM Insufficient memory exists to initialize the thread attributes object.

These functions shall not return an error code of [EINTR].

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the function itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, ``stack size'' is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread cre?

ation.

Attributes objects can be used to set up ``classes' of threads with similar attributes; for example, ``threads with large stacks and high priority" or ``threads with minimal stacks". These classes can be de? fined in a single place and then referenced wherever threads need to be created. Changes to ``class" decisions become straightforward, and de? tailed analysis of each pthread_create() call is not required.

The attributes objects are defined as opaque types as an aid to exten? sibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for im? proving performance. Argument validity can be checked once when at? tributes are set, rather than each time a thread is created. Implemen? tations often need to cache kernel objects that are expensive to cre? ate. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, im? plementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementa? tion.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-in? clusive OR of flags to the initialization routines (pthread_cre? ate(), pthread_mutex_init(), pthread_cond_init()). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify imple? mentation-defined flag values and associated behavior.

2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the pthread_mutex_t and pthread_cond_t objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector at? tributes objects using explicitly-coded bitwise-inclusive OR opera? tions. If the set of options exceeds an int, application program? mers need to know the location of each bit. If bits are set or read by encapsulation (that is, get and set functions), then the bitmask is merely an implementation of attributes objects as currently de? fined and should not be exposed to the programmer.
2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function pa? rameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by get and set functions (making the bit? mask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontiguous and can be al? located and released on demand.

The attribute mechanism has been designed in large measure for extensi? bility. Future extensions to the attribute mechanism or to any at? tributes object defined in this volume of POSIX.1?2017 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as malloc(), may have their size fixed at compile time.

This means, for example, a pthread_create() in an implementation with extensions to pthread_attr_t cannot look beyond the area that the bi? nary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as pos? sibly version information, if extensions in different directions (pos? sibly by different vendors) are to be accommodated.

If an implementation detects that the value specified by the attr argu? ment to pthread_attr_destroy() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the attr argu? ment to pthread_attr_init() refers to an already initialized thread at? tributes object, it is recommended that the function should fail and report an [EBUSY] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_getstacksize(), pthread_attr_getdetachstate(), pthread_create()

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

COPYRIGHT

nel.org/doc/man-pages/reporting_bugs.html .