NAME

libthr — 1:1 POSIX threads library

LIBRARY

1:1 Threading Library (libthr, −lthr)

SYNOPSIS

```
#include <pthread.h>
```

DESCRIPTION

The libthr library provides a 1:1 implementation of the pthread(3) library interfaces for application threading. It has been optimized for use by applications expecting system scope thread semantics, and can provide significant performance improvements compared to N:M Threading Library (libkse, −lkse).

The library is tightly integrated with the run-time link editor ld-elf.so.1(1) and Standard C Library (libc, −lc); all three components must be built from the same source tree. Mixing libc and libthr libraries from different versions of FreeBSD is not supported. The run-time linker ld-elf.so.1(1) has some code to ensure backward-compatibility with older versions of libthr.

The man page documents the quirks and tunables of the libthr. When linking with -lpthread, the run-time dependency libthr.so.3 is recorded in the produced object.

MUTEX ACQUISITION

A locked mutex (see pthread_mutex_lock(3)) is represented by a volatile variable of type lwpid_t, which records the global system identifier of the thread owning the lock. libthr performs a contested mutex acquisition in three stages, each of which is more resource-consuming than the previous. The first two stages are only applied for a mutex of PTHREAD_MUTEX_ADAPTIVE_NP type and PTHREAD_PRIO_NONE protocol (see pthread_mutexattr(3)).

First, on SMP systems, a spin loop is performed, where the library attempts to acquire the lock by atomic(9) operations. The loop count is controlled by the LIBPTHREAD_SPINLOOPS environment variable, with a default value of 2000.

If the spin loop was unable to acquire the mutex, a yield loop is executed, performing the same atomic(9) acquisition attempts as the spin loop, but each attempt is followed by a yield of the CPU time of the thread using the sched_yield(2) syscall. By default, the yield loop is not executed. This is controlled by the LIBPTHREAD_YIELDLOOPS environment variable.

If both the spin and yield loops failed to acquire the lock, the thread is taken off the CPU and put to sleep in the kernel with the _umtx_op(2) syscall. The kernel wakes up a thread and hands the ownership of the lock to the woken thread when the lock becomes available.

THREAD STACKS

Each thread is provided with a private user-mode stack area used by the C runtime. The size of the main (initial) thread stack is set by the kernel, and is controlled by the RLIMIT_STACK process resource limit (see getrlimit(2)).

By default, the main thread's stack size is equal to the value of RLIMIT_STACK for the process. If the LIBPTHREAD_SPLITSTACK_MAIN environment variable is present in the process environment (its value does not matter), the main thread's stack is reduced to 4MB on 64bit architectures, and to 2MB on 32bit architectures, when the threading library is initialized. The rest of the address space area which has been reserved by the kernel for the initial process stack is used for non-initial thread stacks in this case. The presence of the LIBPTHREAD_BIGSTACK_MAIN environment variable overrides LIBPTHREAD_SPLITSTACK_MAIN; it is kept for backward-compatibility.

The size of stacks for threads created by the process at run-time with the `pthread_create`(3) call is controlled by thread attributes: see `pthread_attr`(3), in particular, the `pthread_attr_setstacksize`(3), `pthread_attr_setguardsize`(3) and `pthread_attr_setstackaddr`(3) functions. If no attributes for the thread stack size are specified, the default non-initial thread stack size is 2MB for 64bit architectures, and 1MB for 32bit architectures.

RUN-TIME SETTINGS

The following environment variables are recognized by `libthr` and adjust the operation of the library at run-time:

`LIBPTHREAD_BIGSTACK_MAIN` Disables the reduction of the initial thread stack enabled by `LIBPTHREAD_SPLITSTACK_MAIN`.

`LIBPTHREAD_SPLITSTACK_MAIN` Causes a reduction of the initial thread stack, as described in the section **THREAD STACKS**. This was the default behaviour of `libthr` before FreeBSD 11.0.

`LIBPTHREAD_SPINLOOPS` The integer value of the variable overrides the default count of iterations in the `spin loop` of the mutex acquisition. The default count is 2000, set by the `MUTEX_ADAPTIVE_SPINS` constant in the `libthr` sources.

`LIBPTHREAD_YIELDLOOPS` A non-zero integer value enables the yield loop in the process of the mutex acquisition. The value is the count of loop operations.

`LIBPTHREAD_QUEUE_FIFO` The integer value of the variable specifies how often blocked threads are inserted at the head of the sleep queue, instead of its tail. Bigger values reduce the frequency of the FIFO discipline. The value must be between 0 and 255.

The following `sysctl` MIBs affect the operation of the library:

`kern.ipc.umtx_vnode_persistent` By default, a shared lock backed by a mapped file in memory is automatically destroyed on the last unmap of the corresponding file's page, which is allowed by POSIX. Setting the sysctl to 1 makes such a shared lock object persist until the vnode is recycled by the Virtual File System. Note that in case file is not opened and not mapped, the kernel might recycle it at any moment, making this sysctl less useful than it sounds.

`kern.ipc.umtx_max_robust` The maximal number of robust mutexes allowed for one thread. The kernel will not unlock more mutexes than specified, see `_umtx_op` for more details. The default value is large enough for most useful applications.

`debug.umtx.robust_faults_verbose` A non zero value makes kernel emit some diagnostic when the robust mutexes unlock was prematurely aborted after detecting some inconsistency, as a measure to prevent memory corruption.

The `RLIMIT_UMTXP` limit (see `getrlimit`(2)) defines how many shared locks a given user may create simultaneously.

INTERACTION WITH RUN-TIME LINKER

On load, `libthr` installs interposing handlers into the hooks exported by `libc`. The interposers provide real locking implementation instead of the stubs for single-threaded processes in , cancellation support and some modifications to the signal operations.

`libthr` cannot be unloaded; the `dlclose`(3) function does not perform any action when called with a handle for `libthr`. One of the reasons is that the internal interposing of `libc` functions cannot be undone.

SIGNALS

The implementation interposes the user-installed `signal`(3) handlers. This interposing is done to postpone signal delivery to threads which entered (libthr-internal) critical sections, where the calling of the user-provided signal handler is unsafe. An example of such a situation is owning the internal library lock. When a signal is delivered while the signal handler cannot be safely called, the call is postponed and performed until after the exit from the critical section. This should be taken into account when interpreting `ktrace`(1) logs.

SEE ALSO

`ktrace`(1), `ld-elf.so.1`(1), `getrlimit`(2), `errno`(2), `thr_exit`(2), `thr_kill`(2), `thr_kill2`(2), `thr_new`(2), `thr_self`(2), `thr_set_name`(2), `_umtx_op`(2), `dlclose`(3), `dlopen`(3), `getenv`(3), `pthread_attr`(3), `pthread_attr_setstacksize`(3), `pthread_create`(3), `signal`(3), `atomic`(9)

AUTHORS

The `libthr` library was originally created by Jeff Roberson <`jeff@FreeBSD.org`>, and enhanced by Jonathan Mini <`mini@FreeBSD.org`> and Mike Makonnen <`mtm@FreeBSD.org`>. It has been substantially rewritten and optimized by David Xu <`davidxu@FreeBSD.org`>.