NAME

_lwp_create – create a new light-weight process

SYNOPSIS

#include <sys/lwp.h>

int _lwp_create(ucontext_t **contextp*, **unsigned long** *flags*, **lwpid_t** **new_lwp*);

DESCRIPTION

The function **_lwp_create( )** adds a lightweight process (LWP) to the current process. The *context* parameter specifies the initial signal mask, stack, and machine context (including the program counter and stack pointer) for the new LWP. The new LWP inherits the scheduling class and priority of the caller.

If **_lwp_create( )** is successful, the ID of the new LWP is stored in the location pointed to by *new_lwp*.

flags specifies additional attributes for the new LWP. The value in *flags* is constructed by the bit-wise inclusive OR of the following values:

LWP_DETACHED The LWP is created detached.

LWP_SUSPENDED The LWP is created suspended. If **__LWP_ASLWP** is specified, then the LWP created is the special, designated LWP which handles signals sent to a multi-threaded process — the **aslwp.** There can be only one **aslwp** in a multi-threaded process, so the creation of another **aslwp** will return an error code — **EINVAL.**

__LWP_ASLWP The LWP created is the **aslwp** (Asynchronous Signals LWP) (see **signal**(5)). The **aslwp** should always have all signals blocked — that is how it should be created. It should never exit via **_lwp_exit**(2) or **exit**(2). This flag should not be used by any user program. It is documented here purely for the sake of documentation and not for use by an application.

If **LWP_DETACHED** is specified, then the LWP is created in the *detached* state. Otherwise the LWP is created in the undetached state. The ID (and system resources) associated with a detached LWP can be automatically reclaimed when the LWP exits. The ID of an undetached LWP cannot be reclaimed until it exits and another LWP has reported its termination via **_lwp_wait**(2). This allows the waiting LWP to determine that the waited for LWP has terminated and to reclaim any process resources that it was using.

If **LWP_SUSPENDED** is specified, then the LWP is created in a suspended state. This allows the creator to change the LWP's inherited attributes before it starts to execute. The suspended LWP can only be resumed via **_lwp_continue**(2). If **LWP_SUSPENDED** is not specified the LWP can begin to run immediately after it has been created.

RETURN VALUES

Zero is returned when successful. A non-zero value indicates an error.

ERRORS

If any of the following conditions are detected, **_lwp_create( )** fails and returns the corresponding value:

EFAULT Either the *context* parameter or the *new_lwp* parameter point to invalid addresses.

EAGAIN A system limit is exceeded, e.g., too many LWP were created for this real user ID.

EINVAL The **__LWP_ASLWP** flag was used to create more than one **aslwp** in the process. *There can be only one aslwp within a process.*

EXAMPLES

This example shows how a stack is allocated to a new LWP. **_lwp_makecontext( )** is used to set up the *context* parameter so that the new LWP begins executing a function.

```
contextp = (ucontext_t *)malloc(sizeof(ucontext_t));
stackbase = malloc(stacksize);
sigprocmask(SIGSETMASK, NULL, &contextp->uc_sigmask);
_lwp_makecontext(contextp, func, arg, private, stackbase, stacksize);
error = _lwp_create(contextp, NULL, &new_lwp);
```

SEE ALSO

_lwp_continue(2), **_lwp_exit**(2), **_lwp_makecontext**(2), **_lwp_wait**(2), **exit**(2), **signal**(5), **ucontext**(5)