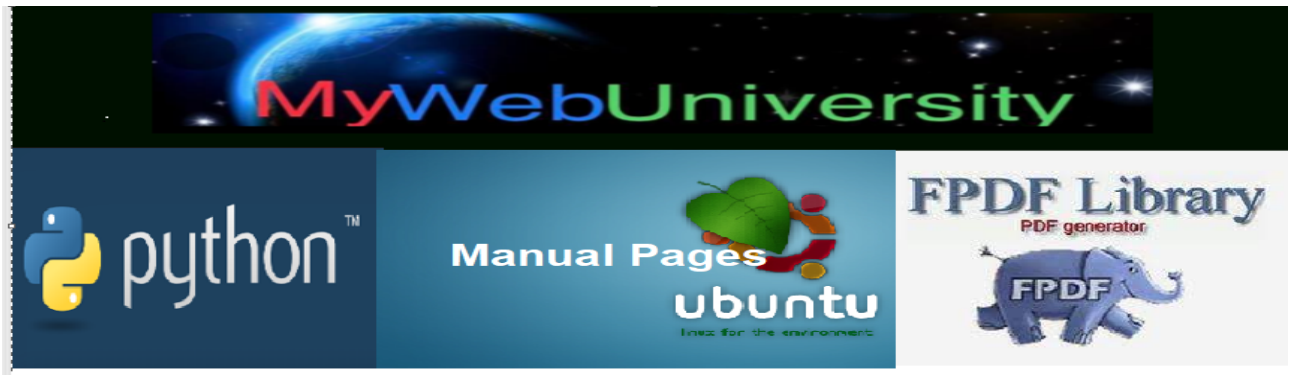

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'pthread_setspecific.3'

$ man pthread_setspecific.3

pthread_key_create(3) Library Functions Manual pthread_key_create(3)

NAME

pthread_key_create, pthread_key_delete, pthread_setspecific, pthread_getspecific - manage? ment of thread-specific data

SYNOPSIS

```
#include <pthread.h>
int pthread_key_create(pthread_key_t *key,
                       typeof(void (void *)) *destr_function;
int pthread_key_delete(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *pointer);
void * pthread_getspecific(pthread_key_t key);
```

DESCRIPTION

Programs often need global or static variables that have different values in different threads. Since threads share one memory space, this cannot be achieved with regular vari? ables. Thread-specific data is the POSIX threads answer to this need.

Each thread possesses a private memory block, the thread-specific data area, or TSD area for short. This area is indexed by TSD keys. The TSD area associates values of type void * to TSD keys. TSD keys are common to all threads, but the value associated with a given TSD key can be different in each thread.

For concreteness, the TSD areas can be viewed as arrays of void * pointers, TSD keys as in? teger indices into these arrays, and the value of a TSD key as the value of the correspond? ing array element in the calling thread.

When a thread is created, its TSD area initially associates NULL with all keys.

pthread_key_create() allocates a new TSD key. The key is stored in the location pointed to by key. There is a limit of PTHREAD_KEYS_MAX on the number of keys allocated at a given time. The value initially associated with the returned key is NULL in all currently execut? ing threads.

The destr_function argument, if not NULL, specifies a destructor function associated with the key. When a thread terminates via pthread_exit() or by cancelation, destr_function is called with arguments the value associated with the key in that thread. The destr_function is not called if that value is NULL. The order in which destructor functions are called at thread termination time is unspecified.

Before the destructor function is called, the NULL value is associated with the key in the current thread. A destructor function might, however, re-associate non-NULL values to that key or some other key. To deal with this, if after all the destructors have been called for all non-NULL values, there are still some non-NULL values with associated destructors, then the process is repeated. The glibc implementation stops the process after PTHREAD_DESTRUC? TOR_ITERATIONS iterations, even if some non-NULL values with associated descriptors remain. Other implementations may loop indefinitely.

pthread_key_delete() deallocates a TSD key. It does not check whether non-NULL values are associated with that key in the currently executing threads, nor call the destructor func? tion associated with the key.

pthread_setspecific() changes the value associated with key in the calling thread, storing the given pointer instead.

pthread_getspecific() returns the value currently associated with key in the calling thread.

RETURN VALUE

pthread_key_create(), pthread_key_delete(), and pthread_setspecific() return 0 on success and a non-zero error code on failure. If successful, pthread_key_create() stores the newly allocated key in the location pointed to by its key argument.

pthread_getspecific() returns the value associated with key on success, and NULL on error.

ERRORS

pthread_key_create() returns the following error code on error:

EAGAIN PTHREAD_KEYS_MAX keys are already allocated.

pthread_key_delete() and pthread_setspecific() return the following error code on error:

EINVAL key is not a valid, allocated TSD key.

pthread_getspecific() returns NULL if key is not a valid, allocated TSD key.

SEE ALSO

pthread_create(3), pthread_exit(3), pthread_testcancel(3).

EXAMPLE

The following code fragment allocates a thread-specific array of 100 characters, with automatic reclamation at thread exit:

```
/* Key for the thread-specific buffer */
static pthread_key_t buffer_key;
/* Once-only initialisation of the key */
static pthread_once_t buffer_key_once = PTHREAD_ONCE_INIT;
/* Allocate the thread-specific buffer */
void buffer_alloc(void)
{
  pthread_once(&buffer_key_once, buffer_key_alloc);
  pthread_setspecific(buffer_key, malloc(100));
}
/* Return the thread-specific buffer */
char * get_buffer(void)
{
  return (char *) pthread_getspecific(buffer_key);
}
/* Allocate the key */
static void buffer_key_alloc()
{
  pthread_key_create(&buffer_key, buffer_destroy);
}
/* Free the thread-specific buffer */
static void buffer_destroy(void * buf)
{
  free(buf);
}
```