线程私有数据可以理解为保存在线程上下文中的数据，我们可以在线程的任何时刻存入数据和取出数据，各个线程私有数据相互隔离

下面接口所需头文件：#include <pthread.h>

**1）创建线程私有数据key**

int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

功能：

创建一个类型为 pthread\_key\_t 类型的私有数据变量( key )。

参数：

**key**：

用于保存返回的key

**destructor**：

清理函数名字。当线程退出时，如果线程私有数据地址不是非 NULL，此函数会自动被调用。该函数指针可以设成 NULL ，这样系统将调用默认的清理函数。其传入的参数为数据的指针

返回值：

成功：0

失败：非 0

**2）注销线程私有数据key**

int pthread\_key\_delete(pthread\_key\_t key);

功能：

注销线程私有数据key

参数：

**key**：待注销的私有数据key

返回值：

成功：0

失败：非 0

**3）设置线程私有数据**

int pthread\_setspecific(pthread\_key\_t key, const void \*value);

功能：

设置线程私有数据( key ) 和 value 指针关联

参数：

**key**：线程私有数据。

**value**：指向数据的指针。

返回值：

成功：0

失败：非 0

**4）读取线程私有数据所关联的值**

**void \*pthread\_getspecific(pthread\_key\_t key);**

功能：

读取线程私有数据key所关联的值。

参数：

**key**：线程私有数据key

返回值：

成功：线程私有数据指针

失败：NULL

示例代码如下：

// this is the test code for pthread\_key

#include <stdio.h>

#include <pthread.h>

#include <sys/types.h>

#include <unistd.h>

#include <iostream>

// 线程私有数据 key

pthread\_key\_t key;

// 线程私有数据销毁函数，当线程退出时会被调用

void echomsg(void \*t)

{

    std::cout << "param = " << \*static\_cast<int \*>(t) << std::endl;

}

// 线程1

void \*child1(void \*arg)

{

    static int i = 10;

pthread\_setspecific(key, &i); // 设置当前线程私有数据

    return nullptr;

}

void \*child2(void \*arg)

{

    static int i = 20;

    // 设置当前线程私有数据

pthread\_setspecific(key, &i);

sleep(1);

    // 获取当前线程私有数据

std::cout << "value = " << \*static\_cast<int\*>(pthread\_getspecific(key)) << std::endl;

    return nullptr;

}

int main(void)

{

    pthread\_t tid1, tid2;

pthread\_key\_create(&key, echomsg); // 创建私有数据 key

    // 创建线程

    pthread\_create(&tid1, NULL, child1, NULL);

pthread\_create(&tid2, NULL, child2, NULL);

    // 等待线程结束

    pthread\_join(tid1, NULL);

pthread\_join(tid2, NULL);

    // 删除私有数据 key

pthread\_key\_delete(key);

    return 0;

}

从运行结果来看，各线程对自己的私有数据操作互不影响。也就是说，虽然 key 是同名且全局，但访问的内存空间并不是同一个。