互斥量本质是一把锁，在访问公共资源前对互斥量设置（加锁），确保同一时间只有一个线程访问数据，在访问完成后再释放（解锁）互斥量。在互斥量加锁之后，其他线程试图对该互斥量再次加锁时都会被阻塞，知道当前线程释放互斥锁。如果释放互斥量时有一个以上的互斥量，那么所有在该互斥量上阻塞的线程都会变成可运行状态，第一个变成运行的线程可以对互斥量加锁，其他线程看到互斥量依然是锁着的，只能再次阻塞等待该互斥量。

互斥量用pthread_mutex_t数据类型表示，在使用互斥量之前，必须使用pthread_mutex_init函数对它进行初始化，注意，使用完毕后需调用pthread_mutex_destroy。

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
    // 两个函数返回值，成功返回0，否则返回错误码
```

pthread_mutex_init用于初始化互斥锁，mutexattr用于指定互斥锁的属性，若为NULL，则表示默认属性。除了用这个函数初始化互斥所外，还可以用如下方式初始化：pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER。

pthread_mutex_destroy用于销毁互斥锁，以释放占用的内核资源，销毁一个已经加锁的互斥锁将导致不可预期的后果。

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
    // 成功返回0，否则返回错误码
```

pthread_mutex_lock以原子操作给一个互斥锁加锁。如果目标互斥锁已经被加锁，则pthread_mutex_lock则被阻塞，直到该互斥锁占有者把它给解锁。

pthread_mutex_trylock和pthread_mutex_lock类似，不过它始终立即返回，而不论被操作的互斥锁是否加锁，是pthread_mutex_lock的非阻塞版本。当目标互斥锁未被加锁时，pthread_mutex_trylock进行加锁操作；否则将返回EBUSY错误码。注意：这里讨论的pthread_mutex_lock和pthread_mutex_trylock是针对普通锁而言的，对于其他类型的锁，这两个加锁函数会有不同的行为。

pthread_mutex_unlock以原子操作方式给一个互斥锁进行解锁操作。如果此时有其他线程正在等待这个互斥锁，则这些线程中的一个将获得它。

`pthread_mutex_trylock` 的应用场景：

例如 格式化 `sd` 卡，你不能使用 pthread_mutex_lock 这样就阻塞了，等前面格式完后，又执行一遍，应该使用 `pthread_mutex_trylock` ,判断返回值，返回值是 `EBUSY` 就可以直接返回 正在格式化了