C++ 读写锁

任心愿

pthread_rwlock_t 读写锁函数说明

读写锁

索引：

1. 初始化一个读写锁 pthread_rwlock_init
2. 读锁定读写锁 pthread_rwlock_rdlock
3. 非阻塞读锁定 pthread_rwlock_tryrdlock
4. 写锁定读写锁 pthread_rwlock_wrlock
5. 非阻塞写锁定 pthread_rwlock_trywrlock
6. 解锁读写锁 pthread_rwlock_unlock
7. 释放读写锁 pthread_rwlock_destroy

具有强读者同步和强写者同步两种形式：

强读者同步：当写者没有进行写操作，读者就可以访问；

强写者同步：当所有写者都写完之后，才能进行读操作，读者需要最新的信息，一些实时性较高的系统可能会用到该所，比如定票之类的。

读写锁使用规则：1、只要没有写模式下加锁，任意线程都可以进行读模式下的加锁；2、只有读写锁处于不加锁状态时，才能进行写模式下的加锁。

读写锁的属性设置

```
[cpp] view plain copy
```

```
1.  /* 初始化读写锁属性对象 */
2.  int pthread_rwlockattr_init (pthread_rwlockattr_t *__attr);
3.
4.  /* 销毁读写锁属性对象 */
5.  int pthread_rwlockattr_destroy (pthread_rwlockattr_t *__attr);
6.
7.  /* 获取读写锁属性对象在进程间共享与否的标识*/
8.  int pthread_rwlockattr_getpshared (__const pthread_rwlockattr_t * __restrict __attr,
9.                                      int *__restrict __pshared);
10.
11. /* 设置读写锁属性对象，标识在进程间共享与否 */
12. int pthread_rwlockattr_setpshared (pthread_rwlockattr_t *__attr, int __pshared);
```

读写锁的使用

[cpp] view plain copy

```
1.  /* 读模式下加锁 */
2.  int pthread_rwlock_rdlock (pthread_rwlock_t *__rwlock);
3.
4.  /* 非阻塞的读模式下加锁 */
5.  int pthread_rwlock_tryrdlock (pthread_rwlock_t *__rwlock);
6.
7.  # ifdef __USE_XOPEN2K
8.  /* 限时等待的读模式加锁 */
9.  int pthread_rwlock_timedrdlock (pthread_rwlock_t *__restrict __rwlock,
10.                               __const struct timespec *__restrict __abstime);
11. # endif
12.
13. /* 写模式下加锁 */
14. int pthread_rwlock_wrlock (pthread_rwlock_t *__rwlock);
15.
16. /* 非阻塞的写模式下加锁 */
17. int pthread_rwlock_trywrlock (pthread_rwlock_t *__rwlock);
```

```
18.
19.  # ifdef __USE_XOPEN2K
20.  /* 限时等待的写模式加锁 */
21.  int pthread_rwlock_timedwrlock (pthread_rwlock_t *__restrict __rwlock,
22.                                  __const struct timespec *__restrict __abstime);
23.  # endif
24.
25.  /* 解锁 */
26.  int pthread_rwlock_unlock (pthread_rwlock_t *__rwlock);
```

（1）pthread_rwlock_rdlock()系列函数

pthread_rwlock_rdlock()用于以读模式即共享模式获取读写锁，如果读写锁已经被某个线程以写模式占用，那么调用线程就被阻塞。在实现读写锁的时候可以对共享模式下锁的数量进行限制（目前不知如何限制）。

pthread_rwlock_tryrdlock()和 pthread_rwlock_rdlock()的唯一区别就是，在无法获取读写锁的时候，调用线程不会阻塞，会立即返回，并返回错误代码 EBUSY。

pthread_rwlock_timedrdlock()是限时等待读模式加锁，时间参数 struct timespec *__restrict __abstime 也是绝对时间，和条件变量的 pthread_cond_timedwait()使用基本一致，具体可以参考 pthread_cond_timedwait() 3 条件变量的使用

（2）pthread_rwlock_wrlock()系列函数

pthread_rwlock_wrlock()用于写模式即独占模式获取读写锁，如果读写锁已经被其他线程占用，不论是以共享模式还是独占模式占用，调用线程都会进入阻塞状态。

pthread_rwlock_trywrlock()在无法获取读写锁的时候，调用线程不会进入睡眠，会立即返回，并返回错误代码 EBUSY。

pthread_rwlock_timedwrlock()是限时等待写模式加锁，也和条件变量的 pthread_cond_timedwait()使用基本一致，具体可以参考 pthread_cond_timedwait() 3 条件变量的使用。

（3）pthread_rwlock_unlock()

无论以共享模式还是独占模式获得的读写锁，都可以通过调用 pthread_rwlock_unlock()函数进行释放该读写锁。

总结（转）：

互斥锁与读写锁的区别：

当访问临界区资源时（访问的含义包括所有的操作：读和写），需要上互斥锁；

当对数据（互斥锁中的临界区资源）进行读取时，需要上读取锁，当对数据进行写入时，需要上写入锁。

读写锁的优点：

对于读数据比修改数据频繁的应用，用读写锁代替互斥锁可以提高效率。因为使用互斥锁时，即使是读出数据（相当于操作临界区资源）都要上互斥锁，而采用**读写锁，则可以在任一时刻允许多个读出者存在，提高了更高的并发度，同时在某个写入者修改数据期间保护该数据，以免任何其它读出者或写入者的干扰。**

读写锁描述：

获取一个读写锁用于读称为共享锁，获取一个读写锁用于写称为独占锁，因此这种对于某个给定资源的共享访问也称为共享-独占上锁。

有关这种类型问题（多个读出者和一个写入者）的其它说法有读出者与写入者问题以及多读出者-单写入者锁。