# 读写锁基本原理

多个线程同时“读”访问共享资源并不会导致问题，在对数据的读写操作中，更多的是读操作，写操作较少。

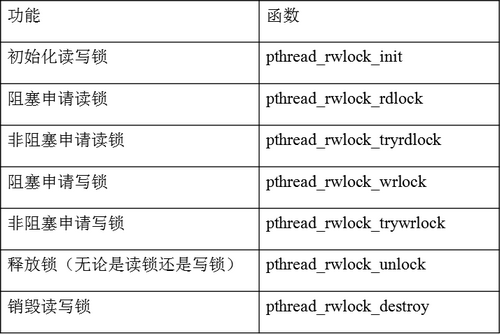
读写锁分为读锁和写锁，规则如下：

1）如果某线程申请了读锁，其它线程可以再申请读锁，但不能申请写锁。

2）如果某线程申请了写锁，其它线程不能申请读锁，也不能申请写锁。

POSIX 定义的读写锁的数据类型是： **pthread\_rwlock\_t**。

# 读写锁基本操作



以下函数所需头文件：

#include <pthread.h>

**初始化读写锁**

int pthread\_rwlock\_init(pthread\_rwlock\_t \*rwlock, const pthread\_rwlockattr\_t \*attr);

功能：

用来初始化 rwlock 所指向的读写锁。

参数：

**rwlock**：指向要初始化的读写锁指针。

**attr**：读写锁的属性指针。如果 attr 为 NULL 则会使用默认的属性初始化读写锁

返回值：

成功：0，读写锁的状态将成为已初始化和已解锁。

失败：非 0 错误码。

**申请读锁**

int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock );

功能：

以阻塞方式在读写锁上获取读锁（读锁定）。

参数：

**rwlock**：读写锁指针。

返回值：

成功：0

失败：非 0 错误码

**试图申请读锁**

int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

用于尝试以非阻塞的方式来在读写锁上获取读锁。 如果有任何的写者持有该锁或有写者阻塞在该读写锁上，则立即失败返回。

**申请写锁**

int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock );

功能：

阻塞方式在读写锁上获取写锁（写锁定）。如果没有“写者”和“读者”持有该锁，则调用线程会获取写锁

参数：

**rwlock**：读写锁指针。

返回值：

成功：0

失败：非 0 错误码

**试图申请写锁**

int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlock);

用于尝试以非阻塞的方式来在读写锁上获取写锁。如果有任何的读者或写者持有该锁，则立即失败返回。

**解锁**

int pthread\_rwlock\_unlock (pthread\_rwlock\_t \*rwlock);

功能：

无论是读锁或写锁，都可以通过此函数解锁。

参数：

**rwlock**：读写锁指针。

返回值：

成功：0

失败：非 0 错误码

**销毁读写锁**

int pthread\_rwlock\_destroy(pthread\_rwlock\_t \*rwlock);

功能：

用于销毁一个读写锁，并释放所有相关联的资源

参数：

**rwlock**：读写锁指针。

返回值：

成功：0

失败：非 0 错误码

# 读写锁应用实例

下面是一个使用读写锁来实现 4 个线程读写一段数据是实例。在此示例程序中，共创建了 4 个线程，其中两个线程用来写入数据，两个线程用来读取数据。当某个线程读操作时，其他线程允许读操作，却不允许写操作；当某个线程写操作时，其它线程都不允许读或写操作。

示例代码如下：

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

pthread\_rwlock\_t rwlock; //读写锁

int num = 1;

//读操作，其他线程允许读操作，却不允许写操作

void \*fun1(void \*arg)

{

    while(1)

    {

        pthread\_rwlock\_rdlock(&rwlock);

        printf("read num first===%d\n",num);

        pthread\_rwlock\_unlock(&rwlock);

        sleep(1);

    }

}

//读操作，其他线程允许读操作，却不允许写操作

void \*fun2(void \*arg)

{

    while(1)

    {

        pthread\_rwlock\_rdlock(&rwlock);

        printf("read num second===%d\n",num);

        pthread\_rwlock\_unlock(&rwlock);

        sleep(2);

    }

}

//写操作，其它线程都不允许读或写操作

void \*fun3(void \*arg)

{

    while(1)

    {

        pthread\_rwlock\_wrlock(&rwlock);

        num++;

        printf("write thread first\n");

        pthread\_rwlock\_unlock(&rwlock);

        sleep(2);

    }

}

//写操作，其它线程都不允许读或写操作

void \*fun4(void \*arg)

{

    while(1)

    {

        pthread\_rwlock\_wrlock(&rwlock);

        num++;

        printf("write thread second\n");

        pthread\_rwlock\_unlock(&rwlock);

        sleep(1);

    }

}

int main()

{

    pthread\_t ptd1, ptd2, ptd3, ptd4;

    pthread\_rwlock\_init(&rwlock, NULL);//初始化一个读写锁

    //创建线程

    pthread\_create(&ptd1, NULL, fun1, NULL);

    pthread\_create(&ptd2, NULL, fun2, NULL);

    pthread\_create(&ptd3, NULL, fun3, NULL);

    pthread\_create(&ptd4, NULL, fun4, NULL);

    //等待线程结束，回收其资源

    pthread\_join(ptd1,NULL);

    pthread\_join(ptd2,NULL);

    pthread\_join(ptd3,NULL);

    pthread\_join(ptd4,NULL);

    pthread\_rwlock\_destroy(&rwlock);//销毁读写锁

    return 0;

}

运行结果如下：

