加餐 - 读者写者问题与读写锁

课堂测试代码：https://gitee.com/whb-helloworld/linux-plus-meal/tree/master/reader_writer_lock

读者写者问题［选学］

读者写者问题

- 重点是 是什么

读者写者 vs 生产消费

- 重点是有什么区别

读者写者问题如何理解

- 重点理解读者和写者如何完成同步

下面是一段伪代码，帮助我们理解读者写者的逻辑

公共部分

```
C++
uint32_t reader_count = 0;
lock_t count_lock;
lock_t writer_lock;
```

Reader

```
C++
// 加锁
lock(count_lock);
if(reader_count == 0)
    lock(writer_lock);
++reader_count;
unlock(count_lock);

// read;
```

```
//解锁
lock(count_lock);
--reader_count;
if(reader_count == 0)
    unlock(writer_lock);
unlock(count_lock);
```

Writer

```
C++
lock(writer_lock);

// write

unlock(writer_lock);
```

读写锁

在编写多线程的时候，有一种情况是十分常见的。那就是，有些公共数据修改的机会比较少。相比较改写，它们读的机会反而高的多。通常而言，在读的过程中，往往伴随着查找的操作，中间耗时很长。给这种代码段加锁，会极大地降低我们程序的效率。那么有没有一种方法，可以专门处理这种多读少写的情况呢?

有，那就是读写锁。

读写锁的行为

当前锁状态	读锁请求	写锁请求
无锁	可以	可以
读锁	可以	阻塞
写锁	阻塞	阻塞

- 注意：写独占，读共享，读锁优先级高

读写锁接口

设置读写优先

```
C
int pthread_rwlockattr_setkind_np(pthread_rwlockattr_t *attr, int
pref);
/*
pref 共有 3 种选择
```

```
PTHREAD_RWLOCK_PREFER_READER_NP (默认设置) 读者优先，可能会导致写者饥饿情况

PTHREAD_RWLOCK_PREFER_WRITER_NP 写者优先，目前有 BUG，导致表现行为和 PTHREAD_RWLOCK_PREFER_READER_NP 一致

PTHREAD_RWLOCK_PREFER_WRITER_NONRECURSIVE_NP 写者优先，但写者不能递归加锁
*/
```

初始化

```
C
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,const pthread_rwlockattr_t *restrict attr);
```

销毁

```
C
int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
```

加锁和解锁

```
C
int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

读写锁案例:

```
C
#include <iostream>
#include <pthread.h>
#include <unistd.h>
#include <vector>
#include <cstdlib>
#include <ctime>

// 共享资源
int shared_data = 0;

// 读写锁
```

```
pthread_rwlock_t rwlock;

// 读者线程函数
void *Reader(void *arg)
{
    //sleep(1); //读者优先，一旦读者进入&&读者很多，写者基本就很难进入了
    int number = *(int *)arg;
    while (true)
    {
        pthread_rwlock_rdlock(&rwlock); // 读者加锁
        std::cout << "读者-" << number << " 正在读取数据, 数据是: " << shared_data << std::endl;
        sleep(1);                       // 模拟读取操作
        pthread_rwlock_unlock(&rwlock); // 解锁
    }
    delete (int*)arg;
    return NULL;
}

// 写者线程函数
void *Writer(void *arg)
{
    int number = *(int *)arg;
    while (true)
    {
        pthread_rwlock_wrlock(&rwlock); // 写者加锁
        shared_data = rand() % 100;     // 修改共享数据
        std::cout << "写者- " << number << " 正在写入. 新的数据是: " << shared_data << std::endl;
        sleep(2);                       // 模拟写入操作
        pthread_rwlock_unlock(&rwlock); // 解锁
    }
    delete (int*)arg;
    return NULL;
}

int main()
{
    srand(time(nullptr)^getpid());
    pthread_rwlock_init(&rwlock, NULL); // 初始化读写锁

    // 可以更高读写数量配比，观察现象
```

```
    const int reader_num = 2;
    const int writer_num = 2;
    const int total = reader_num + writer_num;
    pthread_t threads[total]; // 假设读者和写者数量相等

    // 创建读者线程
    for (int i = 0; i < reader_num; ++i)
    {
        int *id = new int(i);
        pthread_create(&threads[i], NULL, Reader, id);
    }

    // 创建写者线程
    for (int i = reader_num; i < total; ++i)
    {
        int *id = new int(i - reader_num);
        pthread_create(&threads[i], NULL, Writer, id);
    }

    // 等待所有线程完成
    for (int i = 0; i < total; ++i)
    {
        pthread_join(threads[i], NULL);
    }

    pthread_rwlock_destroy(&rwlock); // 销毁读写锁

    return 0;
}
```

Makefile

```
C
reader_writer_lock_test:reader_writer_lock_test.cc
        g++ -o $@ $^ -lpthread
.PHONY:clean
clean:
        rm -f reader_writer_lock_test
```

部分运行效果

```
C++
$ ./reader_writer_lock_test
```

```
写者- 0 正在写入. 新的数据是: 82
读者-0 正在读取数据, 数据是: 82
写者- 0 正在写入. 新的数据是: 32
读者-0 正在读取数据, 数据是: 32
写者- 0 正在写入. 新的数据是: 30
读者-0 正在读取数据, 数据是: 30
写者- 0 正在写入. 新的数据是: 27
读者-0 正在读取数据, 数据是: 27
写者- 0 正在写入. 新的数据是: 43
读者-0 正在读取数据, 数据是: 43
写者- 0 正在写入. 新的数据是: 47
```

读者优先（Reader-Preference）

在这种策略中，系统会尽可能多地允许多个读者同时访问资源（比如共享文件或数据），而不会优先考虑写者。这意味着当有读者正在读取时，新到达的读者会立即被允许进入读取区，而写者则会被阻塞，直到所有读者都离开读取区。读者优先策略可能会导致写者饥饿（即写者长时间无法获得写入权限），特别是当读者频繁到达时。

写者优先（Writer-Preference）

在这种策略中，系统会优先考虑写者。当写者请求写入权限时，系统会尽快地让写者进入写入区，即使此时有读者正在读取。这通常意味着一旦有写者到达，所有后续的读者都会被阻塞，直到写者完成写入并离开写入区。写者优先策略可以减少写者等待的时间，但可能会导致读者饥饿（即读者长时间无法获得读取权限），特别是当写者频繁到达时。