Pthread实现读写锁

1. 内容简介：

编写Pthread程序，使用两个条件变量和一个互斥量来实现一个读写锁。比较当读优先级更高时和写优先级更高时程序的性能。并进行归纳总结。

2. 实现思路：

2.1 读写锁：

读写锁是一种特殊的锁，它把对资源的访问者划分成读者和写者，读者只对共享资源进行读访问，写者则需要对共享资源进行写操作。

一次只有一个线程可以占有写模式的读写锁，但是可以有多个线程占有读模式的读写锁。因此：

- 当读写锁是写加锁的状态时，在这个锁被解锁之前，所有试图对这个锁加锁的线程都会被阻塞；
- 当读写锁在读加锁状态时，所有试图以读模式对它进行加锁的线程都可以获得锁，但是所有试图以写模式对这个锁加锁的线程都会被阻塞；

读优先和写优先：

- 读优先：给读者优先权，只要读写锁不是写加锁的状态，就可以进行读加锁，同时，当一个写线程解锁时，它应该优先唤醒所有的读加锁等待的线程；
- 写优先：给写者优先权，只有当没有读写锁不是写加锁并且没有写等待时，读加锁才能成功，否则就要等待，同时，当一个写线程解锁时，它应该优先唤醒一个写加锁等待的线程；

2) 实现读写锁相关的数据结构：

- 两个条件变量：分别阻塞读线程和写线程；
- 两个读相关的变量，分别记录正在读取的线程数和正在等待读的线程数；
- 两个写相关的变量，分别记录是否有正在写的线程和正在等待写的线程数；
- 一个互斥量，互斥线程对于上述数据的访问和修改；

把这些变量打包成结构体就得到自己实现的读写锁的数据结构：

```
// self-definition read-write-lock
struct my_rwlock_t{

   // use to lock itself
   pthread_mutex_t mutex;

   // read conditional lock
   pthread_cond_t  read;

   //  write conditional lock
   pthread_cond_t  write;

   // record read and write threads
   int read_now;
   int read_wait;
   int write_now;
   int write_wait;
};
```

3. 相关函数：

3.1 读写锁初始化函数：

如图，初始化读写锁的条件变量，互斥锁和相关变量：

```
// initial read-write-lock
void my_rwlock_init(my_rwlock_t* rwlock){
   pthread_mutex_init(&(rwlock->mutex), NULL);

   pthread_cond_init(&(rwlock->read), NULL);

   pthread_cond_init(&(rwlock->write), NULL);

   rwlock->read_now = 0;
   rwlock->read_wait = 0;
   rwlock->write_now = 0;
   rwlock->write_wait = 0;
};
```

3.2 读写锁销毁函数：

销毁读写锁的条件变量和互斥量：

```
// destroy read-write-lock
void my_rwlock_destroy(my_rwlock_t* rwlock){
   pthread_mutex_destroy(&(rwlock->mutex));

   pthread_cond_destroy(&(rwlock->read));

   pthread_cond_destroy(&(rwlock->write));
};
```

3.3 读加锁函数：

根据是否定义了写优先，读加锁有不同的行为：读优先时，只要不是写加锁的状态，就可以进行读加锁；写优先时，只有没有写加锁和写等待才可以进行读加锁：

```
// read lock
void my_rwlock_rdlock(my_rwlock_t* rwlock){

   pthread_mutex_lock(&(rwlock->mutex));

#   ifdef WRITE_FIRST
   // write first
   // if it is writing or some writer is waiting
   if(rwlock->write_wait > 0 || rwlock->write_now > 0) {
      rwlock->read_wait = rwlock->read_wait + 1;
      // wait here
      pthread_cond_wait(&(rwlock->read), &(rwlock->mutex));

      // wake up
      rwlock->read_wait = rwlock->read_wait - 1;
      rwlock->read_now = rwlock->read_now + 1;
   }
   else rwlock->read_now = rwlock->read_now + 1;
#   else
   // read first
   // if no write now
   if(rwlock->write_now == 0) rwlock->read_now = rwlock->read_now + 1;
   // writing now, this thread have to wait
   else{
      rwlock->read_wait = rwlock->read_wait + 1;

      // wait here
      pthread_cond_wait(&(rwlock->read), &(rwlock->mutex));

      // wake up
      rwlock->read_wait = rwlock->read_wait - 1;
```

```
        rwlock->read_now = rwlock->read_now + 1;
    }
#   endif
    pthread_mutex_unlock(&(rwlock->mutex));
};
```

3.4 写加锁函数：

只有当没有读加锁和没有写加锁时，当前线程才可以进行写加锁，否则或被阻塞；

```
// write lock
void my_rwlock_wrlock(my_rwlock_t* rwlock){
    pthread_mutex_lock(&(rwlock->mutex));

    // no read and no write
    if(rwlock->read_now == 0 && rwlock->write_now == 0) rwlock->write_now
= rwlock->write_now + 1;

    // reading or writing
    else{
        rwlock->write_wait = rwlock->write_wait + 1;

        // wait here
        pthread_cond_wait(&(rwlock->write), &(rwlock->mutex));

        // wake up
        rwlock->write_wait = rwlock->write_wait - 1;
        rwlock->write_now = rwlock->write_now + 1;
    }
    pthread_mutex_unlock(&(rwlock->mutex));
};
```

3.5 解锁函数：

根据解锁的线程之前进行的读加锁或者写加锁的不同有不同的行为，根据读优先或者写优先的不同也有不同的行为：

对于读解锁，如果当前读的线程多于1，则将当前正在读的线程数减一即可，如果当前的锁是最后一个读锁，则需要唤醒一个写等待锁（如果有写等待）；

对于写锁，如果读优先，则优先唤醒所有读等待，否则优先唤醒一个的写等待（如果有写等待）；

```
```

```
void my_rwlock_unlock(my_rwlock_t* rwlock){
   pthread_mutex_lock(&(rwlock->mutex));

   // if it is a read thread and there are many readers.
   if(rwlock->read_now > 1) rwlock->read_now = rwlock->read_now - 1;

   // if it is a read thread and there is only one reader.
   else if (rwlock->read_now == 1){
      rwlock->read_now = rwlock->read_now - 1;

      // wake up a writer if needed
      if(rwlock->write_wait > 0) pthread_cond_signal(&(rwlock->write));
   }

   // if it is a writer
   else{
      rwlock->write_now = rwlock->write_now - 1;

      # ifndef WRITE_FIRST
      // read first
      if(rwlock->read_wait > 0) pthread_cond_broadcast(&(rwlock->read));

      else if(rwlock->write_wait > 0) pthread_cond_signal(&rwlock->write);
      #else
      // write first
      if(rwlock->write_wait > 0) pthread_cond_signal(&rwlock->write);
      else if(rwlock->read_wait > 0) pthread_cond_broadcast(&(rwlock-
>read));

      # endif
   }

   pthread_mutex_unlock(&(rwlock->mutex));
};
```

4. 实验结果：

4.1 说明：

- 运行平台：Windows 10，线程数：8；
- 进行测试的是链表程序，有些线程进行查找操作，有些线程进行添加和删除节点操作；

4.2 测试结果（在验证了程序和读写锁的正确性的情况下进行测试，运行时间为s）：

1. 20000次操作，80%读，20%写：

pthread库：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次	
6.882381e-002	6.982398e-002	7.483006e-002	7.180500e-002	7.182717e-002	7.483697e-002	7.280397e-002	7.180810e-002	7

读优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
8.378100e-002	8.678603e-002	7.979298e-002	7.979298e-002	8.378410e-002	8.578920e-002	8.179617e-002	8.079410e-002

写优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次	
7.387710e-002	7.686400e-002	7.488680e-002	7.485700e-002	7.420397e-002	7.683706e-002	7.383609e-002	7.292604e-002	7

2. 20000次操作，50%读，50%写

pthread库：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
1.834972e-001	1.927881e-001	1.935549e-001	1.978250e-001	2.053339e-001	1.966860e-001	1.991560e-001	1.998801e-001

读优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
1.615980e-001	1.586831e-001	1.686320e-001	1.596661e-001	1.566060e-001	1.616001e-001	1.556091e-001	1.616070e-001

写优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
1.456459e-001	1.508100e-001	1.426520e-001	1.426289e-001	1.416628e-001	1.506159e-001	1.476040e-001	1.466429e-001

3. 20000次操作，20%读，80%写
   pthread库：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
3.189881e-001	3.247981e-001	3.329892e-001	3.219309e-001	3.257079e-001	3.262441e-001	3.220551e-001	3.234720e-001

读优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
2.343910e-001	2.381940e-001	2.384188e-001	2.324369e-001	2.344048e-001	2.414820e-001	2.414820e-001	2.513330e-001

写优先：

第一次	第二次	第三次	第四次	第五次	第六次	第七次	第八次
2.171960e-001	2.115428e-001	2.155709e-001	2.185600e-001	2.226849e-001	2.202990e-001	2.244449e-001	2.221761e-001

4.3 结果总结：

- 当写的比例上升时，程序的运行时间显著增加；
- 当读的比例较高时，pthread库提供的读写锁比我自己实现的读优先快，和自己实现的写优先基本一致；但是当写的比例上升时，pthread库的读写锁明显慢于自己实现的读写锁；
- 对比读优先和写优先，不管读写的比例为多少，写优先的策略都稍微快于读优先；

4.4 结果分析：

- 当写的比例上升时，由于写必须互斥（串行执行），程序的运行时间显著增加；
- 写优先快于读优先，是因为读优先使得读的时候有比较多的读线程同时进行，从而使得整个程序的运行时间较短。