Pthreads实现读写锁

17342006 丁聿宁

作业内容：

编写一个Pthreads程序，使用两个条件变量和一个互斥量来实现一个读写锁。下载使用Pthreads读写锁的在线链表程序，修改该程序，让它使用你的读写锁。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。

实现思路：

首先定义读写锁的结构体`my_pthread_rwlock_t`：

```
struct my_pthread_rwlock_t{
    pthread_mutex_t mutex;    //互斥量
    pthread_cond_t read;      //读条件变量
    pthread_cond_t write;     //写条件变量
    int read_wait;            //读者等待数量
    int read_run;             //读者运行数量
    int write_wait;           //写者等待数量
    int write_run;            //写者运行数量
};
```

这里记录读写者的运行和等待数目是为了在解锁的时候知道解的是读锁还是写锁，以及接下来要唤醒的是读者还是写者；使用互斥量是用于保护读写锁的数据结构。

读写锁的初始化函数`my_pthread_rwlock_init`：

```
void my_pthread_rwlock_init(struct my_pthread_rwlock_t* rwlock, void* ptr)
{
    pthread_mutex_init(&(rwlock->mutex), ptr);
    pthread_cond_init(&(rwlock->read), ptr);
    pthread_cond_init(&(rwlock->read), ptr);
    rwlock->read_wait = 0;
    rwlock->read_run = 0;
    rwlock->write_wait = 0;
    rwlock->write_run = 0;
}
```

读写锁的析构函数`my_pthread_rwlock_destroy`：

```
void my_pthread_rwlock_destroy(struct my_pthread_rwlock_t* rwlock)
{
   pthread_mutex_destroy(&(rwlock->mutex));
   pthread_cond_destroy(&(rwlock->read));
   pthread_cond_destroy(&(rwlock->read));
}
```

当线程调用读锁、写锁或解锁函数时，它必须首先锁互斥量，并且无论任何一个线程完成了这些函数调用中的一个，它必须解锁互斥量。在获得互斥量后，线程检查合适的数据成员来决定接下来干什么。无论如何，要满足以下的规定：

- 读读允许
- 读写互斥
- 写写互斥

对于读锁和解锁函数，还要分以下情况讨论：

1.读者优先（强读者同步Strong reader synchronization）：总是给读者优先权，只要写者当前没有进行写操作，读者就能获得访问权。这种情况存在于读者很多，写者不经常更新的时候使用，如图书馆参考数据库采用读者优先比较好。

2.写者优先（强写者同步Strong writer synchronization）：通常把优先权交给写者，而将读者延迟到所有等待的或活动的写者都完成了为止。这种情况存在于经常更新的系统，而读者对最新信息感兴趣，如机票预定系统，写者进行机票价格的更新，读者获取当前机票价。

线程想要进行读访问，就检查是否有一个写者当前拥有锁。如果是写者优先，当有写者在等待或是运行时，读者要被挂起。如果是读者优先，先判断有没有写者在运行，有就被挂起，没有就运行。

如果读者被挂起了，那么它被唤醒后就不等待了，而是在运行。在挂起和运行时，要修改读者运行数目和读者等待数目。

读锁函数`my_pthread_rwlock_rdlock`代码如下：

```
void my_pthread_rwlock_rdlock(struct my_pthread_rwlock_t* rwlock)
{
   pthread_mutex_lock(&(rwlock->mutex));
   if(WRI_PRIOR == 1)
   {
       // 写优先，有写者存在时读者要被挂起
      if(rwlock->write_wait > 0 || rwlock->write_run > 0)
      {
         // 读者等待
         rwlock->read_wait = rwlock->read_wait + 1;
         // 挂起读者
         pthread_cond_wait(&(rwlock->read), &(rwlock->mutex));
         // 读者唤醒后进入运行状态
         rwlock->read_wait = rwlock->read_wait - 1;
         rwlock->read_run = rwlock->read_run + 1;
      }
```

```
        // 没有写者存在可以直接读，允许多个读者运行
        else
            rwlock->read_run = rwlock->read_run + 1;
    }
    else
    {
        // 读优先
        // 如果当前没有写者运行，可以让读者先运行
        if(rwlock->write_run == 0)
            rwlock->read_run = rwlock->read_run + 1;
        // 如果当前有写者运行，要先被挂起，等待写者
        else
        {
            // 读者进入等待队列
            rwlock->read_wait = rwlock->read_wait + 1;
            // 挂起读者
            pthread_cond_wait(&(rwlock->read), &(rwlock->mutex));
            // 读者唤醒后进入运行状态
            rwlock->read_wait = rwlock->read_wait - 1;
            rwlock->read_run = rwlock->read_run + 1;
        }
    }
    // 解开互斥锁
    pthread_mutex_unlock(&(rwlock->mutex));
}
```

线程想要进行写访问，要先判断是否有读者或者写者正在运行，如果没有，那么这个线程进行写操作；如果有，那么这个线程就要被挂起，唤醒后停止等待，开始运行。

写锁函数`my_pthread_rwlock_wrlock`代码如下：

```
void my_pthread_rwlock_wrlock(struct my_pthread_rwlock_t* rwlock)
{
    pthread_mutex_lock(&(rwlock->mutex));

// 只有当前没有读者和写者运行时才能让该写者运行
    if(rwlock->write_run == 0 && rwlock->read_run == 0)
        rwlock->write_run = rwlock->write_run + 1;
// 读写互斥，写写互斥
    else
    {
        rwlock->write_wait = rwlock->write_wait + 1;
        pthread_cond_wait(&(rwlock->write), &(rwlock->mutex));
        rwlock->write_wait = rwlock->write_wait - 1;
        rwlock->write_run = rwlock->write_run + 1;
    }

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

解锁函数的操作取决于线程是一个读者还是一个写者。如果线程是一个读者，且没有其他的读者在运行，并且一个写者正在等待，那么它就在返回前发送信号唤醒写者，使写者继续后继的操作。另一方面，如果线程是写者，则可能同时有读者和写者正在等待。如果是写者优先，则优先选择写者，读者优先，则优先选择读者。

读写锁的解锁函数`my_pthread_rwlock_unlock`代码如下：

```
void my_pthread_rwlock_unlock(struct my_pthread_rwlock_t* rwlock)
{
   pthread_mutex_lock(&(rwlock->mutex));
// 判断锁的类型，读锁
   if(rwlock->read_run > 0)
   {
      // 运行结束
      rwlock->read_run = rwlock->read_run - 1;
      // 如果当前没有读者运行了，看有没有写者在等待
      if(rwlock->read_run == 0)
      {
         // 如果有写者等待，则要唤醒
         if(rwlock->write_wait > 0)
         {
            pthread_cond_signal(&(rwlock->write));
         }
      }
   }

// 写锁
   else if(rwlock->write_run > 0)
   {
      rwlock->write_run = rwlock->write_run - 1;
      // 写优先
      if(WRI_PRIOR == 1)
      {
         if(rwlock->write_wait > 0)
         {
            pthread_cond_signal(&(rwlock->write));
         }
         else if(rwlock->read_wait > 0)
         {
            // 允许多个读者读
            pthread_cond_broadcast(&(rwlock->read));
         }
      }

      // 读优先
      else
      {
         if(rwlock->read_wait > 0)
         {
            pthread_cond_broadcast(&(rwlock->read));
         }
         else if(rwlock->write_wait > 0)
         {
            pthread_cond_signal(&(rwlock->write));
         }
      }
   }
```

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

测试结果

比较当读者优先和写者优先的程序的性能：

8线程，100000次操作，80%查询操作，20%插入操作：

读者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	2.1287	1.9951	2.0360	2.0624	2.0881	2.0335	2.0994	2.0934	2.0804	2.0766

平均运行时间：2.0694s

写者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	2.0125	2.0000	2.0063	1.9980	1.9831	2.0535	2.0038	2.0350	2.0000	2.0945

平均运行时间：2.0186s

8线程，100000次操作，50%查询操作，50%插入操作：

读者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	3.1301	3.6726	3.1544	3.1551	3.1611	3.1047	3.0942	3.1140	3.1006	3.1286

平均运行时间：3.1815s

写者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	3.0983	3.1254	3.0816	3.1436	3.5793	3.1028	3.1365	3.1425	3.1689	3.1369

平均运行时间：3.1715s

8线程，100000次操作，20%查询操作，80%插入操作：

读者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	4.5100	4.3224	4.2564	4.3035	4.3421	4.3208	4.2970	4.3336	4.5815	4.4526

平均运行时间：4.3720s

写者优先：

	1	2	3	4	5	6	7	8	9	10
运行时间(s)	4.4153	4.2578	4.5264	4.4017	4.2720	4.2252	4.4889	4.3336	4.2300	4.2754

平均运行时间：4.3460s

归纳总结

当写操作的比例提高时，读者优先和写者优先运行时间增加。

对比读者优先和写者优先，无论是读操作比例大于写操作，读操作比例等于写操作，还是读操作比例少于写操作的情况，写者优先都稍快于读者优先。