南开大学
Nankai University

南　开　大　学

计 算 机 学 院

并行程序设计实验报告

读写锁算法设计实验报告

周辰霏 1712991

年级：2017 级

专业：计算机科学与技术

2020 年 5 月 6 日

摘要

利用 pthread 其他同步机制（如互斥量、信号量、条件变量等）设计读写锁算法, 并对算法进行相应分析

关键字：pthread；读写锁

目录

一、 概述

(一) 问题描述

读写锁实际是一种特殊的自旋锁，它把对共享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。这种锁相对于自旋锁而言，能提高并发性，因为在多处理器系统中，它允许同时有多个读者来访问共享资源，最大可能的读者数为实际的逻辑 CPU 数。写者是排他性的，一个读写锁同时只能有一个写者或多个读者（与 CPU 数相关），但不能同时既有读者又有写者。[1]

简单概括就是, 同一时间只有一个线程可以占有写者读写锁, 此时其他线程均处于等待阶段; 而同一时间则可以有多个线程同时占有读者读写锁。

(二) 实验内容

针对读写锁进行以下 pthread 同步机制实现:

- pthread 读写锁库函数
- pthread 互斥量、条件变量实现读写锁

并对各种算法进行互斥、前进、等待时间有界分析。

(三) 实验环境

- 系统环境：Windows10 家庭中文版 (64 位)
- 编译环境：gcc 8.1.0 x86_64-posit-seh, Codeblocks17.12, SSE3 指令, pthread

二、 算法设计与实现

(一) POSIX 库函数实现

参考读写锁函数用法 [2] 可以很快写出:

Algorithm 1 POSIX 函数实现

Input: 定义 & 初始化读写锁 rw_lock

function write
    加写锁 $pthread\_rwlock\_wrlock(\&rwlock)$;
    写写写写
    解写锁 $pthread\_rwlock\_unlock(\&rwlock)$;
end function
function read
    加读锁 $pthread\_rwlock\_rdlock(\&rwlock)$;
    读读读读
    解读锁 $pthread\_rwlock\_unlock(\&rwlock)$;
end function

(二) 互斥量实现

根据读写者模型, 实现基本的互斥量读写公平读写者模型。

Algorithm 2 互斥量实现

Input: 定义 & 初始化互斥锁 r_mutex w_mutex rw_mutex, 读者数量 $readers$

```
readers ← 0
function write
    加写锁 pthread_mutex_lock(&w_mutex);
    加读写锁 pthread_mutex_lock(&rw_mutex);
    写写写写
    解读写锁 pthread_mutex_unlock(&rw_mutex);
    解写锁 pthread_mutex_unlock(&w_mutex);
end function
function read
    加写锁 pthread_mutex_lock(&w_mutex);
    加读锁 pthread_mutex_lock(&r_mutex);
    if readers = 0 then
        加读写锁 pthread_mutex_lock(&rw_mutex);
    end if
    readers++
    解读锁 pthread_mutex_unlock(&r_mutex);
    解写锁 pthread_mutex_unlock(&w_mutex);
    读读读读
    加读锁 pthread_mutex_lock(&r_mutex);
    readers--
    if readers = 0 then
        解读写锁 pthread_mutex_unlock(&w_mutex);
    end if
    解读锁 pthread_mutex_unlock(&r_mutex);
end function
```

对算法进行分析, 读写公平的算法可以保证写的时候只有一个进程在临界区 (内存), 而读操作的时候不受限制, 且 r_mutex 可以保证 readers 这一临界资源同时只有一个线程对其访问, 故互斥性满足。对于 progress, 读操作和写操作的等待时间都是有限的, 不会出现等待非参与者的情形。线程选择只会在读写者之间进行。对于 bounded waiting, 读写公平的算法中 rw_mutex 的存在解决了这一问题, 保证了不会有某个线程永远在等待对方操作的情况, 即如果写者先于后续读者到达, 则临界区中读者读完之后就交由写者进行写操作, 等到写完成之后再继续读。

编程实现如下:

rwlock-mutex

```cpp
#include <iostream>
#include <pthread.h>
#include <signal.h>
#include <stdlib.h>
#include<windows.h>
using namespace std;
```

```
#define RN 5 // reader num

pthread_mutex_t wmutex = PTHREAD_MUTEX_INITIALIZER; //临界资源互斥
pthread_mutex_t rmutex = PTHREAD_MUTEX_INITIALIZER; //保护读互斥
pthread_mutex_t rwmutex = PTHREAD_MUTEX_INITIALIZER;//读写互斥

int readers = 0;

void* reader(void *arg)
{
    int id = *(int *)arg;
    while (1)
    {
        Sleep(3);
        pthread_mutex_lock(&rwmutex);
        pthread_mutex_lock(&rmutex);// 一次一个线程修改readers
        readers++;
        if(readers == 1)
            pthread_mutex_lock(&wmutex);
        pthread_mutex_unlock(&rmutex);
        pthread_mutex_unlock(&rwmutex);
        printf("reader %d is reading, current read count[%d]\n", id, readers);
        Sleep(3);
        pthread_mutex_lock(&rmutex);
        readers--;
        if(readers == 0)
            pthread_mutex_unlock(&wmutex);
        pthread_mutex_unlock(&rmutex);
        printf("reader %d is leaving, current read count[%d]\n", id, readers);
    }
    printf("-----reader %d has done ----, current read count[%d]\n", id, readers);
}

void *writer(void *arg)
{
     while(1)
     {
        Sleep(3);
        pthread_mutex_lock(&rwmutex);
        pthread_mutex_lock(&wmutex);
        printf("\twriter is writing\n");
        pthread_mutex_unlock(&rwmutex);// 尽快释放供读者使用，减少同步开销
        pthread_mutex_unlock(&wmutex);
     }
     printf("-----writer has done -----\n");
}

```

```
55 int main(int argc, const char *argv[])
56 {
57     int err;
58     pthread_t tid[RN], writerTid;
59     for(int i=0; i < RN; ++i){
60         err = pthread_create(&tid[i], NULL, reader, &i);
61         if (err != 0)
62         {
63             printf("can't create process for reader");
64         }
65     }
66
67     err = pthread_create(&writerTid, NULL, writer, (void *)NULL);
68     if (err != 0)
69     {
70         printf("can't create process for writer");
71     }
72
73     while(1);
74     return 0;
75 }
```

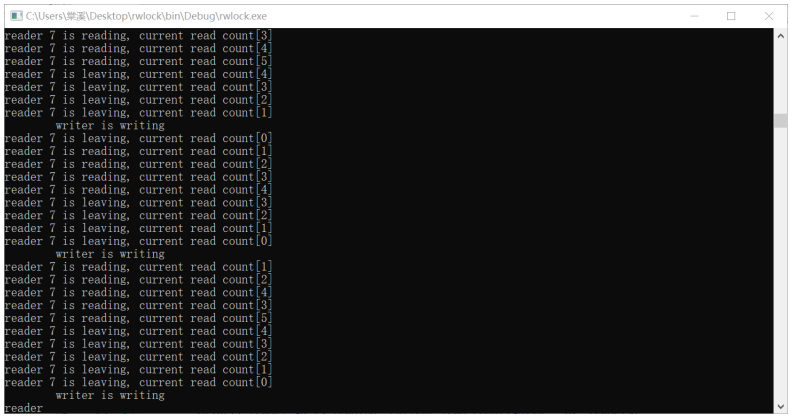

图 1: 测试结果

(三) 信号量实现

过程和互斥量类似。

Algorithm 3 信号量实现

Input: 定义 & 初始化信号量 r_sem w_sem $mutex$, 读者数量 $readers$

1: $readers \leftarrow 0, r\_sem, w\_sem, mutex \leftarrow 1$
2: **function** write
3: $P(mutex)$
4: $P(w\_sem)$
5: 写写写写
6: $V(w\_sem)$

```
    V(mutex)
end function
function read
    P(mutex)
    P(r_sem)
    readers++
    if readers = 1 then
        P(w_sem)
    end if
    V(r_sem)
    V(mutex)
    读读读读
    P(r_sem)
    readers--
    if readers = 0 then
        V(w_sem)
    end if
    V(r_sem)
end function
```

和互斥锁类似, 读写公平的算法多加入了 mutex 这一信号量以及 PV 成对操作保证了互斥性, 即临界区 (内存以及 readers) 只有一个进程使用。对于 progress, 不存在选取是等待非参与者, 进程只会在读写二者中进行选取。对于 bounded waiting, 读写公平的算法中 $mutex$ 的存在解决了这一问题, 如果在读者读的过程中队列中出现了写者, 那么写者加上了对 $mutex$ 进行了 P 操作, 那么当前的读者队列就不会再增加, 直到当前队列读者运行完写者写完之后才可以继续向队列中添加读者, 保证了有限等待。

三、 总结

本次实验算是彻底实现了操作系统课上就学到的读写者模型, 从同步互斥的原理理解到 pthread 的实现, 整体不难, 但是要注意保证临界区的三个条件。

参考文献

[1] 读写锁 [EB/OL].https://baike.baidu.com/item/%E8%AF%BB%E5%86%99%E9%94%81

[2] pthread_rwlock[EB/OL].https://www.daemon-systems.org/man/pthread_rwlock.3.html

[3] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=SSE3,AVX&expand=3931

[4] 课件 algo1-2,openmp10-11,pthread3-6