Improving the fairness of Reader-Writer Solution from Exercise 4

Department of Computer Science
University of Applied Science Darmstadt

Subject: Operating Systems
Winter semester 2019/2020

Maciej Krzyszton and
Christian Kehr (gr. Mi5x)

Email:
maciej.krzyszton.stud.h-da.de
christian.kehr.stud.h-da.de

Introduction

The Reader-Writer problems describe the issues associated with concurrent resources sharing in multithreaded application. The general problem definition tells that one set of data is shared among many threads, reader can only read, writer can only write, if at least one reader is reading and at any given time, only one writer can write, and no reader can read while a writer writes

There are three main known problems which base on this definition, first where reader threads have more operations, second with writer being preferred and the third which says that both readers and writers should be handled fairly.

In this text we focus on reader preferred solution on try to make it fairer. We have prepared two solutions, first one in which only one thread can access shared resources. The second solution allows more threads to access the resources if there is no writing thread at the time. To prove and demonstrate our Implementations we use given to use code for database simulation[1] where number of read and write operations are executed and measured.

Theory

The first implementation "Reader Preferred" gives exclusive access to database and only one reader can read at the given time. While there is a reader in queue, writing is prohibited. Only if reader counter is zero, writing operation will be allowed.

[1] Thanks to Prof. Dr. Ronald Moore

Solution 1 “Reader-Preferred “

Reader thread

```
resLock.lock();
readerCount++;
if(readerCount==1)
    writeLock.lock();
resLock.unlock();

theDatabase.read(readerID);

resLock.lock();
readerCount--;
if(readerCount==0)
    writeLock.unlock();
resLock.unlock();
```

Writer thread

```
writeLock.lock();
bool result =
theDatabase.write(writerID);
++tests;
writeLock.unlock();
```

Mutale variables

```
int readerCount = 0;
std::mutex resLock,
writeLock;
```

The second fairer implementation’s advantage over previous one is multiple access on read operation and using semaphores(condition_variable). Once there is no writing thread, all waiting writing threads are unblocked. In our understanding fairer means:

1) No thread should starve
2) No writing while reading, nor reading while writing
3) Multiple reader threads can read at the same time

In theory, it should improve overall throughput over “Reader Preferred” solution and reduce overall waiting time of operations.

Solution 2 “Fairer Reader-Writer”

Reader thread

```
resLock.lock();

while (writerActive) {
    cV.wait(ulock);
}
++numReadersActive;
resLock.unlock();

bool result =
theDatabase.read(readerID);

resLock.lock();
--numReadersActive;
if(numReadersActive == 0){
    cV.notify_one();
}
resLock.unlock();
```

Writer thread

```
++numWritersWaiting;
while (writerActive ||
numReadersActive != 0) {
    cV.wait(ulock);
}
writerActive = true;
resLock.unlock();

bool result =
theDatabase.write(writerID);

resLock.lock();
writerActive = false;
--numWritersWaiting;
if(numWritersWaiting>0){
    cV.notify_one();
}
else{
    cV.notify_all();
}
resLock.unlock();
```

Shared variables

```
int numReadersActive = 0;
int numWritersWaiting = 0;
bool writerActive = false;

std::mutex resLock;
std::unique_lock<std::mutex>
ulock(resLock);
std::condition_variable cV;
```

Measurements

We have tested both programs on same machine with twenty threads maximal pro test. In test 1 we have writers and readers thread number balanced, in test 2 readers have the edge and in test 3 writers have the edge. All tests were measured over same time of 500 second (5 minutes and 20 seconds), one after the other. We have measured and compared the number of all operations and average waiting time for completion of the operation, also we checked overall performance difference.

Table 1

Threads pro role	**Statistics**	**Reader-Preferred**	**Fairer Reader-Writer**	**Change over Reader-Preferred**	**Test time**	**Test Num:**
10	Reader num of Op	959	1054	+10%	500s	1
	Avg reader wait time	0.146181	0.0990718	+48%		
10	Writer num of Op	648	636	-2%		
	Avg writer wait time	0.695454	0.648182	+7%		
15	Reader num of Op	2443	2805	+15%	500s	2
	Avg reader wait time	0.147204	0.083362	+76%		
5	Writer num of Op	523	446	-15%		
	Avg writer wait time	0.70205	0.652871	+7%		
5	Reader num of Op	357	326	-9%	500s	3
	Avg reader wait time	0.113702	0.106494	+7%		
15	Writer num of Op	725	732	+1%		
	Avg writer wait time	0.672019	0.649917	+3%		

All test where performed on Macbook Pro Intel Dual Core i5 2.6 GHz, SSD 256GB, 8gb DDR3 RAM 1600Mhz macOS 10.15.1 and Kernel Darwin 19.0.0. All benchmark programs are written in C++17 standard and compiled with Clang. Percentages have been rounded to closer whole number

Table 2

Test Num:	**Sum of operations**	**Reader-Preferred**	**Fairer Reader-Writer**	**Change over Reader-Preferred**
1	Sum of all operations	1607	1690	+5%
	Avg time	0.4208175	0.3736269	+12%
2	Sum of all operations	2966	3251	+10%
	Avg time	0.424627	0,3681165	+14%
3	Sum of all operations	1082	1058	-2%
	Avg time	0.3928605	0.3782055	+4%

Table 3

Statistics	**Reader-Preferred**	**Fairer Reader-Writer**	**Change over Reader-Preferred**
Sum of operations in tests	5655	5999	+6%
Average wait time	0.412768	0,373316	+11%

Evaluation

While looking at the column “change over Reader-Preferred” we notice in all tables for test 1 and 2 mostly improvement. For Reader operation count, average waiting time (improved drastically) and especially when we look in test 2(see table 2) where there were more readers than writers. The new solution makes better use overall of more reader threads (see test 1 and 2 in Table 2) yet, it does not upgrade writer performance all that much, if not worse it.

When we look at the test 3 where there are more writing threads, we notice not a fair exchange between number of read compare to write operations for “Fairer Reader-Writer”. We also do see slightly fewer overall operations in Table 2 test 3 which appears strange. Yet now we think of it like a limitation, which appear when only one thread is allowed to write and assign more threads to writer role does not help it. The change of average waiting by 3% is not that important when we see lose in overall amount of operations. It could be due to OS scheduler.

Summary

Overall results are positive (see Table 3). However, we were not able to improve the number of write operations, yet we have shortened the waiting time in all cases which is significant in test 2 it has been over 76% better for the reader.

Test 3 shows us the limitation when only one thread is allowed to write. More advanced software could keep track of which parts of the database resources are being used and allowing more concurrent writing operations, could definitely improve the “fairer Reader-Writer” in test 3, for now we cannot use full potential.