Operating Systems - CS 304

Rishit Saiya - 180010027, Assignment - 2

March 18, 2021

1 Abstract

There are two flavors of the reader-writer lock, which we will illustrate with an example. Suppose a reader thread R1 has acquired a reader-writer lock for reading. While R1 holds this lock, a writer thread W and another reader thread R2 have both requested the lock. Now, it is fine to allow R2 also to simultaneously acquire the lock with R1, because both are only reading shared data. However, allowing R2 to acquire the lock may prolong the waiting time of the writer thread W, because W has to now wait for both R1 and R2 to release the lock. So, whether we wish to permit more readers to acquire the lock when a writer is waiting is a design decision in the implementation of the lock. When a reader-writer lock is implemented with reader preference, additional readers are allowed to concurrently hold the lock with previous readers, even if a writer thread is waiting for the lock. In contrast, when a reader-writer lock is implemented with writer preference, additional readers are not granted the lock when a writer is already waiting.

2 Implementation

2.1 rwlock.h

The header file `rwlock.h` had to edited to add appropriate CV (Condition Variables), Locks and some variables pertaining to `read_write_lock` structure created.

```
struct read_write_lock{
    int doRead;
    int doWrite;
    int waitNumberofReader;
    int waitNumberOfWriter;
    pthread_cond_t readerCV;
    pthread_cond_t writerCV;
    pthread_mutex_t mutex;
};
```

```
└─$ ./test-rwlock.sh
TESTSET: Running Testcases with reader preference
test-reader-pref.cpp:7:6: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismat
ch]
    7 | long index;
      |      ^~~~~
test-reader-pref.cpp: In function 'void* Reader(void*)':
test-reader-pref.cpp:39:1: warning: no return statement in function returning non-void [-Wreturn-type]
   39 | }
      | ^
test-reader-pref.cpp: In function 'void* Writer(void*)':
test-reader-pref.cpp:64:1: warning: no return statement in function returning non-void [-Wreturn-type]
   64 | }
      | ^
CASE1: Reader Preference with 5 reader and 1 writer
PASSED
CASE2: Reader Preference with 5 reader and 3 writer
PASSED
CASE3: Reader Preference with 5 reader and 5 writer
PASSED
TESTSET: Running Testcases with writer preference
test-writer-pref.cpp:7:6: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismat
ch]
    7 | long index;
      |      ^~~~~
test-writer-pref.cpp: In function 'void* Reader(void*)':
test-writer-pref.cpp:39:1: warning: no return statement in function returning non-void [-Wreturn-type]
   39 | }
      | ^
test-writer-pref.cpp: In function 'void* Writer(void*)':
test-writer-pref.cpp:64:1: warning: no return statement in function returning non-void [-Wreturn-type]
   64 | }
      | ^
CASE1: Writer Preference with 5 reader and 1 writer
PASSED
CASE2: Writer Preference with 5 reader and 3 writer
PASSED
CASE3: Writer Preference with 5 reader and 5 writer
PASSED
Test Cases Passed: 6
Test Cases Total: 6
```

Figure 1: Final Output

2.2 rwlock-reader-pref.cpp & rwlock-writer-pref.cpp

The `rwlock-reader-pref.cpp` is the file pertaining to reader preference and `rwlock-writer-pref.cpp` is the file pertaining to the writer preference. The functions had to be completed using Spin Locks & appropriate additions of CV with variables.

3 Analysis of Various Cases

In the given script `test-rwlock.sh`, various situations with different readers/writers preferences with different number of readers/writers are given. In the following subsections, analysis on different PASSED test cases are explained.

3.1 CASE1: Reader Preference with 5 reader and 1 writer

In CASE1, the default preference is given to Reader and following are cases which can occur in such situations.

3.1.1 Readers Scheduled Before

In the default cases and most of the cases, Readers are scheduled before the writer. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 1 Writer, all the readers are scheduled before writing assuming that writer wasn't already in process. After all the 5 readers have finished the process of reading, they signal the writer to start writing and is scheduled sequentially and written.

3.1.2 Readers Scheduled After

In the some rare occurrence of cases, where the writer process is already locked, the group of 5 readers will have to get added in the queue for scheduling. Once the writer completes the process of writing, it signals all the 5 Readers and then they get scheduled and read concurrently.

3.2 CASE2: Reader Preference with 5 reader and 3 writer

In CASE2, the default preference is given to Reader and following are cases which can occur in such situations.

3.2.1 Readers Scheduled Before

In the default cases and most of the cases, Readers are scheduled before the writers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 3 Writers, all the readers are scheduled before writing assuming that writer wasn't already in process. After all the 5 readers have finished the process of reading, they signal all 3 writers to start writing and are scheduled sequentially. After the Writer 1 finishes writing, Writer 2 is signalled and it starts its process and so on.

3.2.2 Readers Scheduled After

In the some rare occurrence of cases, where the writer process is already locked by Writer 1, the group of 5 readers will have to get added in the queue and wait for scheduling. Once the Writer 1 completes the process of writing, it signals all the 5 Readers and then they get scheduled and read concurrently. Post all the reading processes, sequentially all the writers which were in the queue and processed sequentially and turn wise.

3.3 CASE2: Reader Preference with 5 reader and 3 writer

In CASE2, the default preference is given to Reader and following are cases which can occur in such situations.

3.3.1 Readers Scheduled Before

In the default cases and most of the cases, Readers are scheduled before the writers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 3 Writers, all the readers are scheduled before writing assuming that writer wasn't already in process. After all the 5 readers have finished the process of reading, they signal all 3 writers to start writing and are scheduled sequentially. After the Writer 1 finishes writing, Writer 2 is signalled and it starts its process and so on.

3.3.2 Readers Scheduled After

In the some rare occurrence of cases, where the writer process is already locked by Writer 1, the group of 5 readers will have to get added in the queue and wait for scheduling. Once the Writer 1 completes the process of writing, it signals all the 5 Readers and then they get scheduled and read concurrently. Post all the reading processes, sequentially all the writers which were in the queue and processed sequentially and turn wise.

3.4 CASE3: Reader Preference with 5 reader and 5 writer

In CASE3, the default preference is given to Reader and following are cases which can occur in such situations.

3.4.1 Readers Scheduled Before

In the default cases and most of the cases, Readers are scheduled before the writers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 5 Writers, all the readers are scheduled before writing assuming that writer wasn't already in process. After all the 5 readers have finished the process of reading, they signal all 5 writers to start writing and are scheduled sequentially. After the Writer 1 finishes writing, Writer 2 is signalled and it starts its process and so on.

3.4.2 Readers Scheduled After

In the some rare occurrence of cases, where the writer process is already locked by Writer 1, the group of 5 readers will have to get added in the queue and wait for scheduling. Once the Writer 1 completes the process of writing, it signals all the 5 Readers and then they get scheduled and read concurrently. Post all the reading processes, sequentially all the writers which were in the queue and processed sequentially and turn wise.

3.5 CASE1: Writer Preference with 5 reader and 1 writer

In CASE1, the default preference is given to Writer and following are cases which can occur in such situations.

3.5.1 Writers Scheduled Before

In the default cases and most of the cases, Writers are scheduled before the readers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 1 Writer, the writer is scheduled before reading processes assuming that reader wasn't already in process. After the writer has finished its process, it will signal all the 5 readers and then 5 readers start reading concurrently.

3.5.2 Writers Scheduled After

In the some rare occurrence of cases, where the reader processes are already locked by 5 Readers, the writer will have to get added in the queue and wait for scheduling. Once all the 5 readers processes are finished (concurrently), it signals the writer and then the writer get scheduled.

3.6 CASE2: Writer Preference with 5 reader and 3 writer

In CASE2, the default preference is given to Writer and following are cases which can occur in such situations.

3.6.1 Writers Scheduled Before

In the default cases and most of the cases, Writers are scheduled before the readers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 3 Writer, the writer 1 is scheduled before other writers & reading processes assuming that reader wasn't already in process. After the writer 1 has finished its process, it will signal other writers to execute. Post all sequential executions of writers, all the 5 readers are signalled and then 5 readers start reading concurrently.

3.6.2 Writers Scheduled After

In the some rare occurrence of cases, where the reader processes are already locked by 5 Readers, the 3 writers will have to get added in the queue and wait for scheduling. Once all the 5 readers processes are finished (concurrently), it signals the writers and then the writers get scheduled and finish their executions sequentially and individually in that order.

3.7 CASE3: Writer Preference with 5 reader and 5 writer

In CASE3, the default preference is given to Writer and following are cases which can occur in such situations.

3.7.1 Writers Scheduled Before

In the default cases and most of the cases, Writers are scheduled before the readers. The main idea to be noted is that all the readers can read the content concurrently whilst the writers have to write singularly. Hence in this case of 5 Readers and 5 Writer, the writer 1 is scheduled before other writers & reading processes assuming that reader wasn't already in process. After the writer 1 has finished its process, it will signal other writers to execute. Post all sequential executions of writers, all the 5 readers are signalled and then 5 readers start reading concurrently.

3.7.2 Writers Scheduled After

In the some rare occurrence of cases, where the reader processes are already locked by 5 Readers, the 5 writers will have to get added in the queue and wait for scheduling. Once all the 5 readers processes are finished (concurrently), it signals the writers and then the writers get scheduled and finish their executions sequentially and individually in that order.

4 Conclusion

With all the cases confined above, we have demonstrated the variation of scheduling with controlled preferences over Readers and Writers.