DEPARTMENT OF COMPUTER SCIENCE, HANYANG UNIVERSITY

(DOCUMENT VERSION 1.3)

# Simple Two-phase locking with Readers-writer Lock

**Concurrent Programming**

Programming Project #2

Final due date: Oct 25, 2019 (HARD DEADLINE)

## 1 TASK OVERVIEW

Two-phase locking is one of widely-used concurrency control algorithms in database systems. This assignment is to **implement two-phase locking with readers-writer lock** within a simple structure.

## 2 TASK DETAILS

You have $N$ worker threads and $R$ records in your database. Each record is a 64 bit signed integer (initial value: 100). Each thread executes transactions until the *global execution or-*

*der* (initial value: 0) reaches the value *E*.

Each transaction consists of a sequence of following actions below.

1. Randomly pick up three different records *i*, *j*, *k*, respectively.
2. Acquire a global mutex that protects a lock table.
3. Acquire a reader lock for reading a value of the record *i*, $R_i$.
    - If it need to wait for acquiring the lock, do *deadlock checking*.
4. Release the global mutex.
5. Read $R_i$.
6. Acquire the global mutex again.
7. Acquire a writer lock for writing a value of the record *j*, $R_j$.
    - If it need to wait for acquiring the lock, do *deadlock checking*.
8. Release the global mutex.
9. Increase the value of $R_j$ by 1, i.e., $R_j = R_j + R_i + 1$.
   ($R_i$ is the value you have read at step 5)
10. Acquire the global mutex again.
11. Acquire a writer lock for writing a value of the record *k*, $R_k$.
    - If it need to wait for acquiring the lock, do *deadlock checking*.
12. Release the global mutex.
13. Decrease $R_k$ by $R_i$. i.e., $R_k = R_k - R_i$.
    ($R_i$ is the value you have read at step 5)

    ———————— committing phase ————————
14. Acquire the global mutex.
15. Release all reader/writer locks acquired by this transaction.
16. Increase the global execution order by 1, and then fetch it as *commit_id*. Initial value of the global execution order is 0 so that the first *commit_id* need to be 1.

- If *commit_id* is bigger than *E*, rollback all changes made by this transaction (Undo), and then release the global mutex, and terminate the thread.

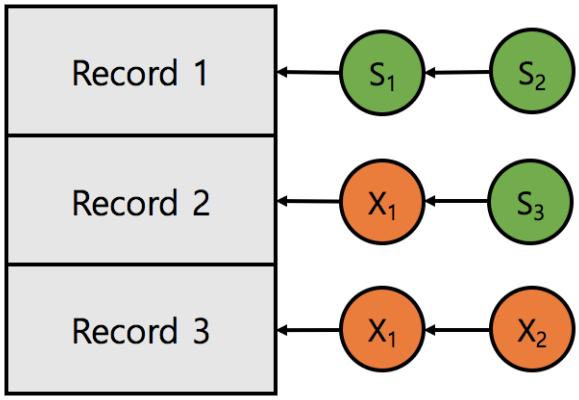
Append a **commit log** into the thread#.txt with the below format:

[*commit_id*] [*i*] [*j*] [*k*] [$R_i$] [$R_j$] [$R_k$]

17. Release the global mutex. ———————— committed ————————

## 3 READERS-WRITER LOCK

Whenever a transaction accesses a record, it must acquire the corresponding record-level lock. You need to implement reader-writer lock for allowing multiple transactions to read the same record at the same time. Lock objects for a single record should be managed in a linked-list structure so that you can keep the order of the transactions accessing the record. When a transaction releases the lock, it must wake up other waiting transaction(s) who are safe to be proceed, if exists.

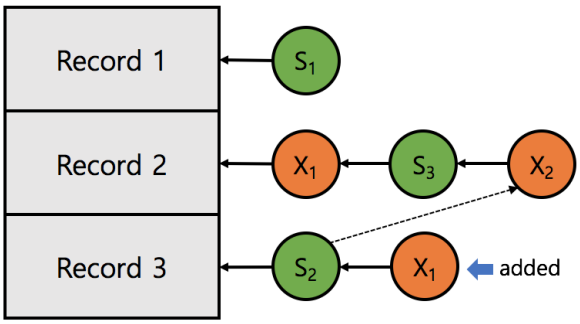



< $S_t$: A reader lock of transaction *t*, $X_t$: A writer lock of transaction *t*>

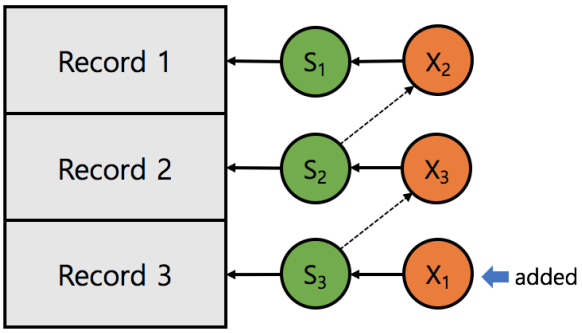
In the figure above, transaction 1 acquired three locks, and did not commit yet. Transaction 2 is acquiring a read lock for record 1, and waiting for acquiring a write lock for record 3. Transaction 3 is waiting for acquiring a read lock for record 2.

# 4 DEADLOCK CHECKING

Whenever a transaction needs to wait for acquiring a reader/writer lock, it must perform a deadlock checking. It can detect a dead lock by detecting a *wait-for* cycle in the lock table. If it turns out to be a deadlock, the transaction must rollback all changes it made (Undo), and restart.




< Deadlock example 1: transaction 1 -> transaction 2 -> transaction 1 >




< Deadlock example 2: transaction -> transaction 3 -> transaction 2 -> transaction 1 >

# 5 TEST PROTOCOL

Your program should take three command line arguments. Example of the execution is:

| $ ./run *N R E* |
|---|
| *N*: Number of worker threads<br>*R*: Number of records<br>*E*: Last global execution order which threads will be used to decide termination |

After your program finished, $N$ output files need to be generated. The name of each output file is thread#.txt (i.e., if $N$ is 3, thread1.txt, thread2.txt, thread3.txt need to be generated.) Each commit log should be printed on each line of the corresponding file.
The format of the output file is below: ($N$=3, $R$=3)

| thread1.txt |
| --- |
| 1 1 2 3 100 201 0<br>2 2 3 1 201 202 -101<br>4 3 1 2 102 2 200<br>... |

| thread2.txt |
| --- |
| 3 1 3 2 -101 102 302<br>... |

| thread3.txt |
| --- |
| 5 2 1 3 200 203 -98<br>... |

## 6 SUBMISSION

You should upload your project into **project2** directory of your "hconnect" repository. You must create a proper **Makefile** in the project2 directory. The name of your executable file must be **run**, and it must be at the project2 directory as well.
You also need to upload your assignment report in your gitlab wiki page of your hconnect project. Please set a name of the wiki page as project2.

**Please enjoy this project and have fun !!**