Concurrent Programming Assignment2, Simple Two-phase locking with Readers-writer Lock  
컴퓨터공학부 성예닮

Prof. 정형수

TAs. 김종빈, 오현석, 장형원

Run:

./run [N] [R] [E]

Ex)   
./run 3 5 100 3

Task

데이터베이스 시스템에서 사용되는 병렬 기법 중 하나인 2PL을 읽기-쓰기 Lock과 함께 구현한다.

Thread 하나는 한 트랜잭션을 반복적으로 수행한다.  
해당 트랜잭션이 하는 일은 다음과 같다.

1. 3개의 서로 다른 i, j, k를 선택한다.
2. 전역 Mutex를 잡는다.
3. Record i를 읽기 위해 읽기 Lock을 잡는다. (Deadlock 발생 여부 확인)
4. 전역 Mutex를 풀어준다.
5. Record i를 읽는다.
6. 전역 Mutex를 잡는다.
7. Record j에 쓰기 위해 쓰기 Lock을 잡는다. (Deadlock 발생 여부 확인)
8. 전역 Mutex를 풀어준다.
9. Record j의 값에 Record i의 값 + 1을 더한다.
10. 전역 Mutex를 잡는다..
11. Record k에 쓰기 위해 쓰기 Lock을 잡는다. (Deadlock 발생 여부 확인)
12. 전역 Mutex를 풀어준다.
13. Record k의 값에 Record i의 값을 뺀다.

데이터베이스에 Commit

1. 전역 Mutex를 잡는다..
2. 현재 트랜잭션에서 잡은 읽기-쓰기 lock을 모두 해제한다
3. Global execution order를 1 증가시키고 Commit id로 해당 값을 받아온다.  
   (Commit id가 Maximum order보다 크다면,  
   Undo 연산 후, 전역 Mutex를 풀고, Thread를 종료)
4. thread#.txt 파일에 Commit log를 기록한다.  
   [*commit\_id*] [*i*] [*j*] [*k*] [*value of record i*] [*value of record j*] [*value of record k*]
5. 전역 Mutex를 해제한다

해당 트랜잭션을 Global execution order가 사용자에게 입력받은 Maximum order보다 커질 때까지 수행한다.

Data structures

rw\_lock\_t: // 읽기-쓰기 Lock의 정보를 가지고 있는 구조체이다.  
lock\_type type\_;  
lock\_state state\_;  
int tid\_ // 해당 Lock을 요청한 thread의 id, 시스템의 tid가 아닌 thread 생성 순서이다.

node\_t: // Lock 테이블의 하나의 노드 구조체이다.  
 rdlock\_cnt; // 해당 노드의 읽기 Lock 개수  
 wrlock\_cnt; // 해당 노드의 쓰기 Lock 개수  
 list <rw\_lock\_t> lock\_list // 해당 노드에서의 Lock list, 첫번째 요소는 반드시 RUNNING중이다.

list <record\_t> records[N]

list <note\_t> lock\_table[N]; // N은 Record 수와 동일하다.

Functions

void\* run\_trx(void \*arg):

트랜잭션을 수행하는 Thread에 인자로 전달되는 함수이다.  
인자로는 Thread의 생성 순서 번호를 전달한다.

void set\_rand\_nums(int n\_record, int \*rand\_nums):  
전체 Record 중 Random한 index를 rand\_nums 배열에 설정한다.

void acquire\_rdlock(rw\_lock\_t lock, int tid, int record):  
읽기 Lock을 요청한다.  
요청하고자 하는 Lock table에 대기중인 Lock이 많거나, 바로 실행할 수 없는 경우 WAITING으로 처리하고 Waiting list에 추가한다.  
Lock table의 record 인덱스에 Lock의 정보와 해당 Lock을 요청한 Thread의 번호를 기록한다.

void acquire\_wrlock(rw\_lock\_t lock, int tid, int record):

쓰기 Lock을 요청한다.  
읽기 Lock의 요청과 다른 점은, 쓰기 Lock을 요청한 후에는 Deadlock cycle을 체크해야 한다는 것이다.

void release\_lock(int tid, int record, lock\_type t)

Lock의 타입을 체크하여 읽기 Lock 혹은 쓰기 Lock을 해제한다.

void release\_trx\_lock(int i, int j, int k, int tid)

현재 트랜잭션에서 잡은 모든 Lock을 해제한다.

void rollback\_records(int i, int j, int k, long long i\_value, long long j\_value, long long k\_value)

트랜잭션이 중간에 실패하거나 order가 꼬인 경우, 트랜잭션에서 변경한 모든 Record를 복구시킨다.

bool deadlock\_checking(int level, int tid, int target, stack<int> &run\_list,  
 int i, int j, int k, int my\_idx)

쓰기 Lock을 요청한 후 Lock table graph에서, Cycle이 발생하면 Deadlock 상태가 된다.  
해당 상태를 확인하고 Undo를 요청한다.

다음과 같은 단계를 거쳐 Cycle을 체크한다.

1. 요청한 Lock이 있는 Record의 Running Lock List을 확인한다.
2. Running Lock List에 1개 이상의 Lock이 존재한다면, 해당 Lock들의 Pair(다른 Record에 있는 같은 Thread의 Lock)에 대해 순차적으로 1번 연산을 진행한다.
3. Stack에 쌓이며 연산을 진행하다 Running Lock List에 처음 시작한 Thread의 tid가 다시 한번 발견되면 Deadlock임을 확인한다.(like DFS)