**멀티코어 프로그래밍**

**Programming Project 4 Report**

**정형수 교수님**

**컴퓨터공학부 컴퓨터전공**

**2013011424 성예닮**

**목차**

**0. 실행 방법**

**1. 과제 목표**

**2. 프로그램 설계**

**3. 코드 설계**

**4. 결과**

**0. 실행 방법**

압축을 해제하고 2013011424/ 에서 make 후 ./bin/homework으로 실행합니다.

실행 시 옵션은 --table\_size --num\_thread --read\_num --duration이 있습니다.

각각 테이블의 사이즈와, 실행할 스레드, 읽기 연산 시 읽을 레코드의 개수, 실행 시간입니다.

기본 값으로는 --table\_size=10000 --num\_thread=synconf(\_SC\_NPROCESSORS)

--read\_num=10 --duration=30 입니다.

sysconf(\_SC\_NPROCESSORS)는 시스템콜 함수로 현재 환경의 코어 수를 반환해주는 함수입니다.

**1. 과제 목표**

Simple & Fast Transaction Locking System

2 phase lock을 사용해서 Transaction locking system을 구현하는 것입니다.

2 phase lock이란, concurrent하게 작업하되 직렬 가능성을 보장해주는 규약입니다.

한번 lock을 하면, 그 다음에는 unlock이 나오기 전까지 lock만 가능하고,

한번 unlock을 하면 쭉 그 다음부터는 unlock만 가능합니다.

이번 과제는 크게 Transaction table, Lock Hash table, Lock list, Record list를 구현하는 것입니다.

Transaction table에서는 각각의 트랜잭션이 취할 행동, record에 대해 READ와 UPDATE를 취하되, 이 부분에서 lock으로 접근하는 record를 잠급니다.

잠근 record는 table id와 record id로 hashing되어 lock hash table의 hash key에 맞는 lock list에 lock으로 잠겨있다고 표시되고, 어떤 transaction이 record를 잠그고 있는지도 표시합니다.

잠긴 lock은 shared와 exclusive 두 종류입니다.

접근하려는 record가 이미 잠겨있을 경우, lock list의 뒤에 lock을 추가하고 기다립니다.

shared의 경우 read lock과 같고 exclusive의 경우 write lock과 동일합니다.

또한 이렇게 lock 잠금에 대해 cycle이 생기는 경우(A가 B을 필요로 하고, B가 A을 필요로 하는데 둘 다 잠겨있는 상황)에는 deadlock check를 통해 abort시키고, A와 B가 가진 lock을 모두 해제합니다.

이렇게 프로그램을 진행해서 몇 회 READ, UPDATE, Transaction, Abort 진행되었는지 출력합니다.

**2. 프로그램 설계**

1. record table 생성

2. hash lock table 생성

3. transaction(thread) table 생성

4. transaction이 할 일

READ opeartion

4-1. global counter로 transaction id를 받아옵니다.

4-2. random\_record\_k, random\_table\_l을 선택합니다.

4-3. random\_table\_l의 random\_record\_k부터 read\_num개 만큼 shared\_lock으로 잠급니다.

잠근다는 것은 random\_table\_l과 random\_record\_k로 hashing한 값을 hash key로 삼아

lock hash table의 자신의 위치에 lock list에 추가하는 것입니다.

4-3-1. 만약 이미 잠겨있다면, lock list의 뒤에 추가하고 해당 lock을 waiting 상태로 변경하고 sleep 합니다.

4-4. random\_k ~ random\_k + read\_num -1을 모두 summation 합니다.

4-5. read count를 1 증가시킵니다.

UPDATE operation

5-1. table 두 개의 random\_record + read\_num ~ random\_record + 9까지 exclusive lock으로 잠급니다.

5-2. 이미 잠겨있다면, lock list의 뒤에 추가하고 해당 lock을 waiting 상태로 변경합니다.

5-3. dead lock cycle을 검사하고, deadlock이라면 abort 시키고 abort count를 1 증가시킵니다.

abort 시 속한 transaction의 모든 lock을 해제합니다.

5-4. table 두 개의 random\_record + read\_num ~ random\_record + 9에 해당하는 record에

5-4-1. 만약 READ가 A table이라면 A table에서는 -10, B table에서는 +10을 하고

5-4-2. 만약 READ가 B table이라면 B table에서는 -10, A table에서는 +10을 합니다.

5-5. update count를 1증가시킵니다.

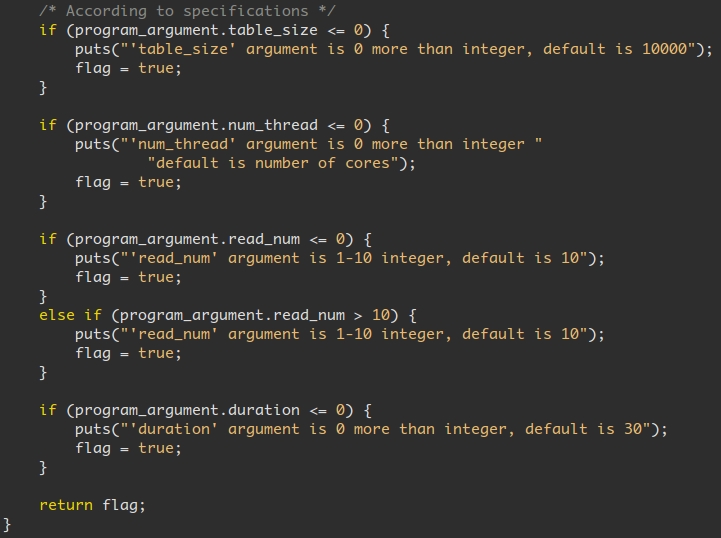
5-6. sleep하고 있는 transaction을 모두 깨우고, transaction count를 1 증가시킵니다.

**3. 코드 설계**



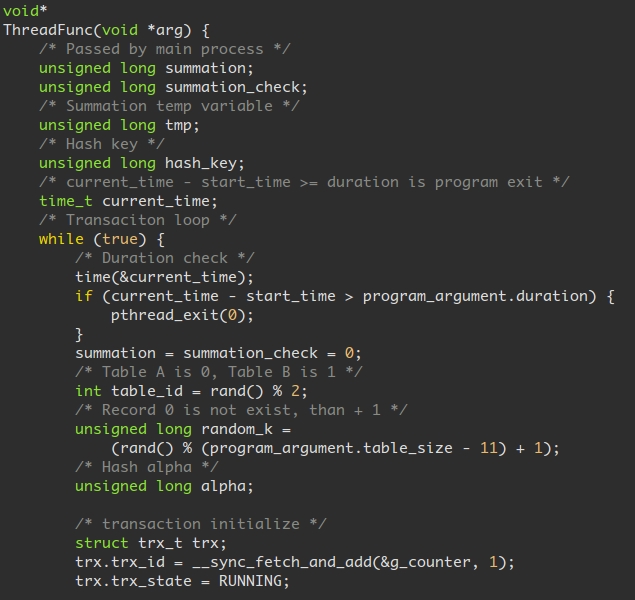
인자로 받은 argv를 parsing하는 함수입니다.

getopt\_long함수를 사용하였고, parsing 실패 시 ture, 성공시 false 를 반환합니다.



parsing 실패 시 나오는 문구입니다.

주어진 명세에 따라 작성되었습니다.

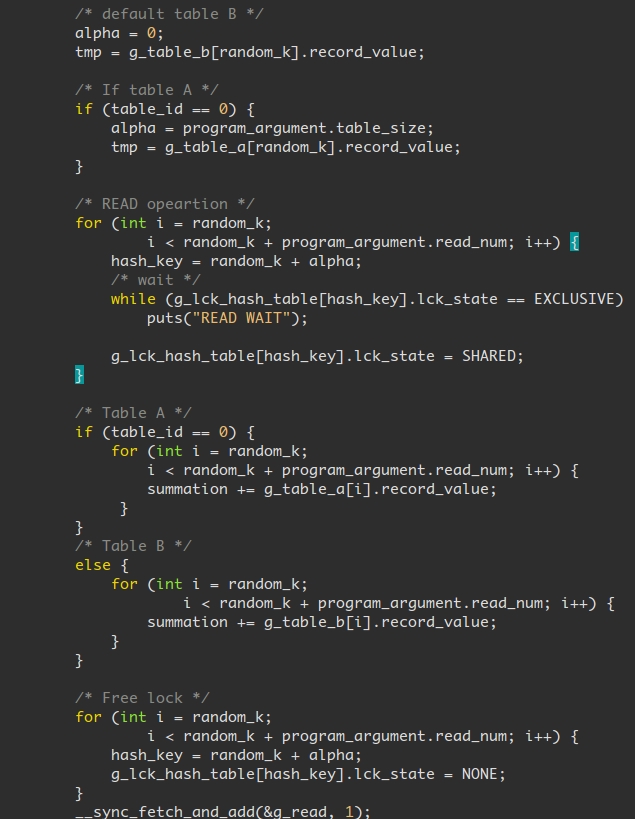


쓰레드 루틴입니다. 트랜잭션 하나의 역할을 하고, 매 루프 마다 시간을 계산합니다.

랜덤 k를 선택하고, hashing용 alpha를 선언합니다.

트랜잭션 아이디를 원자적으로 가져오고, 카운터를 1증가시킵니다.

트랜잭션 상태를 RUNNING으로 표시합니다.

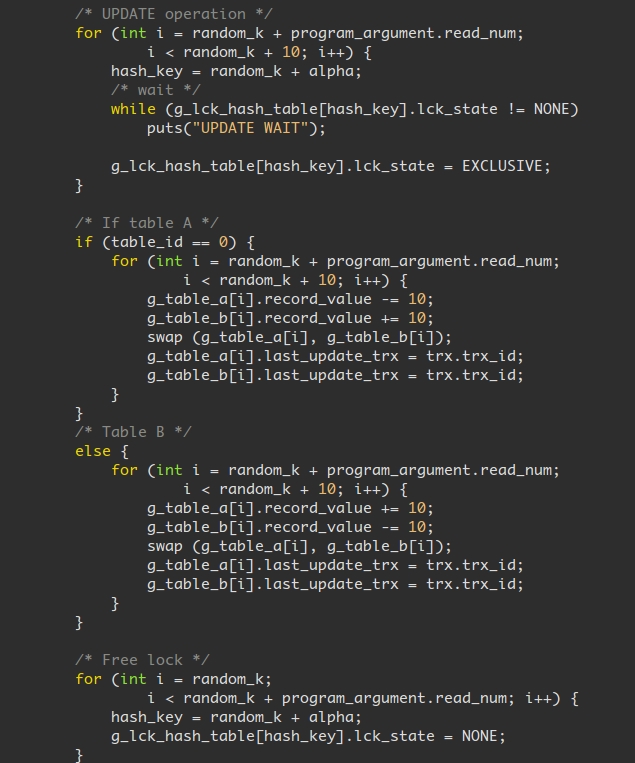


READ 작업입니다.

랜덤한 k부터 read num의 개수까지 루프를 돌며, EXCLUSIVE인 경우 대기를 합니다.

NONE혹은 SHARED라면 SHARED로 표기하고 READ연산을 계속 합니다.

레코드의 합산을 구하고, read 카운터를 하나 증가시킵니다.

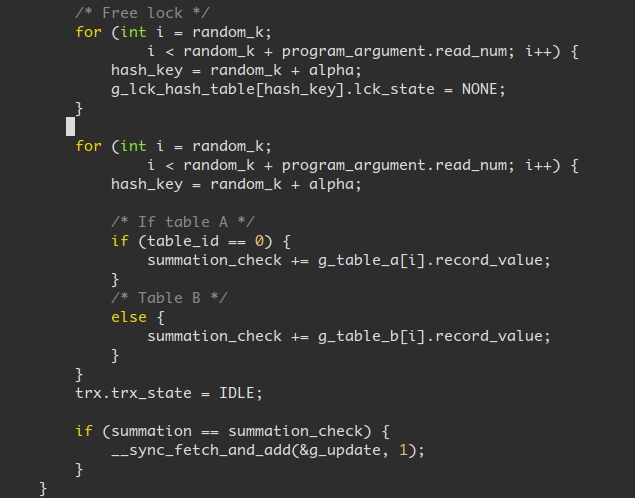


UPDATE 작업입니다.

랜덤한 k + read num부터, 랜덤한 k + 9 까지 루프를 돌며 EXCLUSIVE락을 표시합니다.

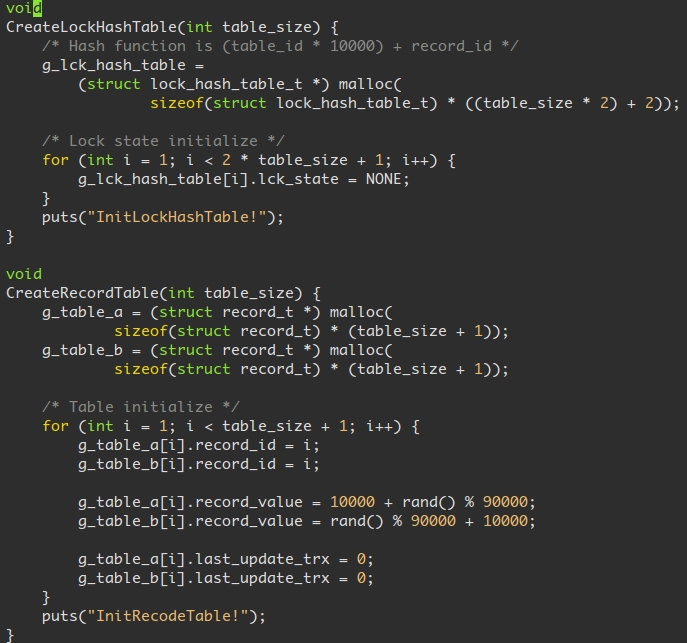
이전에 READ작업에서 테이블 A를 읽었다면, 테이블 A의 레코드 값을 -10, 테이블 B의 레코드 값을 +10 계산하고, 두 수를 바꿉니다.

그리고 마지막 업데이트 트랜잭션으로 현재 트랜잭션 아이디를 표시합니다.



업데이트 락을 해제하고, 이전의 합과 비교하기 위해 다시 합산을 구합니다.

트랜잭션 상태를 IDLE로 표기하고, 이전의 합과 같으면 UPDATE를 1 증가시킵니다.



해시 테이블과 레코드 테이블을 생성하는 함수입니다.

해시 테이블의 경우 락 리스트를 사용하는 방식이 아니라

해시된 값 자체의 테이블이 락을 표시하도록 했습니다.

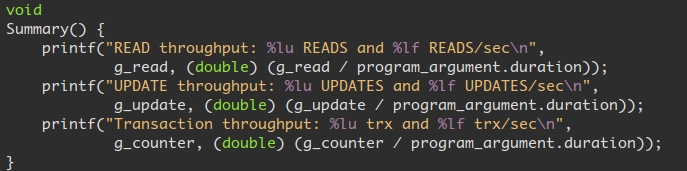
따라서 아래와 같이 해시 테이블에 락 리스트가 연결된 형식이 아니라,

ㅣ ㅡㅡㅡㅡ

ㅣ ㅡㅡㅡㅡ

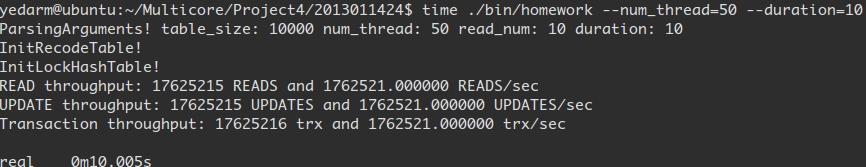
해시 테이블만 존재합니다. (ex ㅡㅡㅡㅡㅡ)

레코드 테이블의 경우 랜덤으로 값을 정해주고, 마지막 트랜잭션 업데이트를 0으로 표기하여 초기화합니다.



Duration 시간만큼 작동하고, 해당 시간 후에 쓰레드를 종료하고 위의 함수를 통해 Throughput을 출력합니다.

**4. 결과**



위와 같은 결과가 나옵니다.

아마 READ opeartion만 정상 작동 하는 것 같습니다………………