Lab2: Concurrent Data Structure

과목명 : 운영체제(SW)
교수명 : 최종무 교수님
학 과 : 소프트웨어학과
학 번 : 32183698
이 름 : 이현기
제출일 : 2022/05/05

DKU 단국대학교 DANKOOK UNIVERSITY

1. 목표

이번 Lab2의 목표는 생산자 소비자 문제에 기초를 둔 다중 쓰레드로 된 프로그램을 만드는 것으로 쓰레드들은 큐와 해시 자료구조를 공유하며, 몇몇 쓰레드들은 큐와 해시의 자료를 삽입하거나 삭제하고 다른 쓰레드들은 해시를 이용하여 검색을 수행한다. 여기서 3가지 비교를 해야한다. 먼저 lock이 있을때와 없을 때, 그리고 fine grained lock과 coarse grained lock의 비교, 그리고 쓰레드 개수를 다르게 하였을 때의 수행능력을 비교한다.

큐는 기본적인 자료구조로 FIFO(First In First Out)방식을 사용한다. Lab2에서는 이중연결리스트로 이루어진 큐를 사용하며, 큐의 맨 앞을 front, 제일 뒤를 rear라 부른다. 큐에다가 값을 넣는 enqueue, 큐에 있는 값을 삭제하는 dequeue연산을 수행한다.

해시테이블은 키와 값으로 데이터를 저장하는 자료구조로 Bucket구조를 통해 빠른 검색을 할 수 있도록 한다. 각 키마다 해시 함수를 적용하여 인덱스를 만들고 이 인덱스를 활용하여 데이터를 저장하고 검색을 한다.

lock은 임계영역에 한번에 한 쓰레드만 접근할 수 있게 만드는 것으로, lock을 하는 크기에 따라 fine grained lock과 coarse grained lock으로 나눌 수 있다. fine grained lock은 작은 단위의 임계영역을 많이 lock하는 것으로 병렬성이 좋다는 장점이 있지만 복잡하다는 단점이 있다. coarse grained lock은 lock의 개수를 작게하여 큰 단위의 임계영역을 lock하는 것으로 간단하다는 장점이 있지만 병렬성이 없다는 단점이 있다. Lab2에서 이러한 3가지 비교를 해보고 각각 시간에 차이가 있는지 살펴본다.

2. 설계/구현

이번 lab2에서 직접 구현해야할 파일은 lab2_sync.c 파일이다. 해시와 큐에 대한 파일이며 큐에 대해서는 큐를 초기화하는 함수, enqueue와 dequeue를 lock없이 할 때, coarse grained lock을 사용할 때, fine grained lock을 사용할 때 이렇게 3가지로 나누어 각각 따로 함수를 작성하였다.

해시에 대해서는 해시리스트의 노드를 초기화하는 함수, 해싱을 하는 함수, 해시 큐

에 넣는 것을 lock이 없을 때, coarse grained lock을 할 때, fine grained lock을 할 때 총 3가지로 따로따로 함수를 작성하였다. 해시테이블에 지정된 값이 있는지 체크하는 함수, 타겟을 이용하여 해시큐에 삽입하는 것과 삭제하는 것을 lock이 없을 때, coarse grained lock을 할 때, fine grained lock을 할 때 총 3가지로 따로따로 함수를 작성하였다.

3. 결과

작성한 코드를 make하여 실행파일을 생성하고 예제에 있던 -t=4 -c=1000000 -l=0 을 수행한 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=4 -c=1000000 -l=0


=====  Multi Threads No-lock HQ  Insert experiment =====

 Experiment Info
        Test Node Count                 : 1000000
        Test Threads                    : 4
        Execution Time                  : 0.00 seconds

=====  Multi Threads No-lock HQ  Delete experiment =====

 Experiment Info
        Test Node Count                 : 1000000
        Test Threads                    : 4
        Execution Time                  : 0.00 seconds

        Total Execution Time            : 0.00 seconds

oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$
```

똑같은 -t=4 -c=1000000를 주고 -l=1로 하였을 때의 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=4 -c=1000000 -l=1


=====  Multi Threads Coarse-grained HQ  Insert experiment =====

 Experiment Info
    Test Node Count         : 1000000
    Test Threads            : 4
    Execution Time          : 0.00 seconds

=====  Multi Threads Coarse-grained HQ  Delete experiment =====

 Experiment Info
    Test Node Count         : 1000000
    Test Threads            : 4
    Execution Time          : 0.00 seconds

  Total Execution Time        : 0.00 seconds
```

이번에도 같은 옵션에 -l=2로 하였을 때의 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=4 -c=1000000 -l=2


===== Multi Threads Fine-grained HQ  Insert experiment =====

 Experiment Info
    Test Node Count         : 1000000
    Test Threads            : 4
    Execution Time          : 0.00 seconds

===== Multi Threads Fine-grained HQ  Delete experiment =====

 Experiment Info
    Test Node Count         : 1000000
    Test Threads            : 4
    Execution Time          : 0.00 seconds

  Total Execution Time        : 0.00 seconds
```

이번에는 -t를 10으로, -c를 100000000로 수정하고 -l=0를 수행한 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=10 -c=100000000 -l=0


=====  Multi Threads No-lock HQ  Insert experiment =====

 Experiment Info
       Test Node Count               : 100000000
       Test Threads                  : 10
       Execution Time                : 4.00 seconds

=====  Multi Threads No-lock HQ  Delete experiment =====

 Experiment Info
       Test Node Count               : 100000000
       Test Threads                  : 10
       Execution Time                : 5.00 seconds

       Total Execution Time          : 9.00 seconds
```

-l=1로 하였을 때 수행 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=10 -c=100000000 -l=1


=====  Multi Threads Coarse-grained HQ  Insert experiment =====

 Experiment Info
    Test Node Count         : 100000000
    Test Threads            : 10
    Execution Time          : 4.00 seconds

=====  Multi Threads Coarse-grained HQ  Delete experiment =====

 Experiment Info
    Test Node Count         : 100000000
    Test Threads            : 10
    Execution Time          : 4.00 seconds

  Total Execution Time        : 8.00 seconds
```

-l=2로 하였을 때의 수행 결과이다.

```
oslab@oslab:~/Desktop/2022_DKU_OS/lab2_sync$ ./lab2_sync -t=10 -c=100000000 -l=2


===== Multi Threads Fine-grained HQ  Insert experiment =====

 Experiment Info
    Test Node Count         : 100000000
    Test Threads            : 10
    Execution Time          : 4.00 seconds

===== Multi Threads Fine-grained HQ  Delete experiment =====

 Experiment Info
    Test Node Count         : 100000000
    Test Threads            : 10
    Execution Time          : 4.00 seconds

  Total Execution Time        : 8.00 seconds
```

4. 논의

먼저 실행을 하기 전에는, lock이 없을 때 임계영역에 대한 문제점이 생겨 실행결과가 제대로 나타나지 않을 것이고, fine-grained lock을 하였을 때 시간이 많이 걸리고 coarse grained lock을 하였을 때 fine grained lock보다 시간이 적게 걸릴 것이라 예상하였다. 예시에 주어진 쓰레드 개수와 노드 카운트를 하였을 때에는 3가지 모두 0초가 나왔다. 쓰레드 개수와 노드 카운트를 증가시켜 하였을 때에는 락이 없을 때 9초, coarse grained lock은 8초, fine grained lock도 8초가 나왔다. 먼저 lock이 없을 때도 실행결과가 나타나서 의외였고, coarse grained lock과 fine grained lock의 시간 차이가 없어서 의외였다. 데이터의 개수를 많이 하면 1~2초 차이가 생기긴 하지만 오차범위안에 드는 차이라고 생각한다. 데이터의 수를 어느정도까지 설정해야 눈에 보이는 차이가 날지 궁금하였고, 많다고 생각한 데이터의 양도 바로 처리하는 것을 보니 하드웨어의 기술이 많이 발전하여 처리하는 속도가 빨라졌다는 것을 느꼈다.

코드를 작성하는 과정에서 해시충돌의 경우를 생각하지 않고 코드를 작성하였다. 그래서 실행에 문제가 있지 않을까 하였는데 문제는 발생하지 않았지만, 해시 충돌에 대해서도 생각하여 코드를 작성하면 좀 더 완벽한 프로그램이 되지 않을까란 생각을 하였다. 그리고 큐에 대해서는 어느정도 알고 있었지만, 해시에 대한 지식이 많이 없어서 해시를 구현하는데 어려움이 있었다. 큐와 해시같은 자료구조에 대해 다시 한번 복습할 필요성을 느꼈다.