**운영체제  
Lab 2. Concurrency**

폰트, 로고, 등록 상표, 상징이(가) 표시된 사진

자동 생성된 설명

주 제 : Linux에서 세 가지 Skip List 구현

교 수 : 최건희

학 과 : 소프트웨어학과

학 번 : 32190789

이 름 : 김승호

마 감 : 25.05.21.

**소스코드 설명**

**1. DefaultSkipList (락 없는 SkipList)**

동기화가 적용되지 않은 순수한 SkipList입니다.

단일 스레드 환경에서만 안전하게 동작합니다.

이미 존재하는 키에 대해 value를 누적하고, upd\_cnt를 증가시킵니다.

**SkipList::insert 코드 설명**

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**SkipList::Lookup 코드 설명**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**SkipList::remove 코드 설명**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2. Coarse-grained Lock**

pthread\_mutex\_t 을 이용해 SkipList 전체를 보호하는 전역 락 방식

하나의 요청을 수행하는 동안 모든 스레드가 차단됨

insert, lookup, remove 모든 작업에서 lock을 획득/반환하도록 구현함

**CoarseSkipList::insert 코드 설명**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**CoarseSkipList::lookup 코드 설명**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**CoarseSkipList::remove 코드 설명**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**3. Fine-grained Lock**

각 노드에 대해 별도의 락 (FineNode::lock)을 적용

탐색 시 다음 노드의 락을 먼저 획득한 후 이전 노드의 락을 해제하는 hand\_over 방식 적용

insert 시 중복 키가 존재하면 값 누적과 upd\_cnt ++수행

delete시 pthread\_mutex\_destroy 를 통해 노드 락 해제 및 메모리 안전 제거 수행

**문제 풀이**

**1. 멀티 스레드 환경에서 SkipList에 접근할 때, 요청 수행 순서를 일관되게 보장하는 방법**

일관된 요청 순서 보장은 **임계영역 보호**와 **정확한 락 획득 순서**에 의해 달성된다. Coarse 방식에서는 단일 전역 락을 통해 모든 작업의 선후 관계가 보장되며, Fine 방식에서는 노드 단위 락을 정해진 순서로 획득하면서 deadlock 없이 탐색-삽입-삭제 순서를 안전하게 유지함. 특히 hand-over 방식으로 락 획득 시, 항상 "앞 노드 락 보유 → 다음 노드 락 획득 → 앞 노드 락 해제" 순서를 유지하면 동시성 충돌을 방지할 수 있다.

**2. Fine-grained SkipList에서 동일한 키에 대해 동시에 insert와 remove가 들어오는 경우**

락이 노드 단위로 적용되므로 insert와 remove는 해당 키가 포함된 노드의 락을 두 스레드가 쟁탈하게 된다. 먼저 락을 획득한 스레드의 작업이 선행되고, 이후 스레드는 상태에 따라 insert 후 새 노드를 만들거나, remove 후 아무 작업도 하지 않게 됨. 결과는 락 획득 순서에 따라 다르게 결정되며, 하나의 스레드가 먼저 insert하여 upd\_cnt++가 발생한 후 다른 스레드가 remove하여 노드 삭제가 일어나는 흐름도 가능함.

**Discussion**

**여러 워크로드에서, lock이 적용되는 범위가 다를 경우, thread의 개수에 따른 실행시간 비교 분석**

1. **Without Lock(기본)**은 단일 스레드에서만 빠르게 동작하며, 멀티 스레드에서는 충돌로 오작동 가능성이 높다.

2. **Coarse** 방식은 병렬성은 거의 없지만 일관성은 확실하다. 하지만 스레드 수가 늘어날수록 락 경쟁으로 인한 병목 현상이 심해짐.

3. **Fine** 방식은 스레드가 많아질수록 병렬 삽입/삭제가 가능하므로 확실한 성능 향상을 보임. 단, 락 관리 로직이 복잡하다.

**과제를 진행하며 새롭게 배운 점/ 어려웠던 점**