과목명	운영체제
담당교수	최종무 교수님
학과	소프트웨어학과
학번	32151671 / 32153180
이름	박민혁 / 이상민
제출일자	2019.05.08

Ⅰ. 프로젝트 개요

1. 프로젝트 목표

본 프로젝트에서는 binary search tree에서 pthread 기반 mutex를 활용해 race condition 상황을 해결한다.

2. 프로젝트 기본 개념

Critical Section & Race Condition

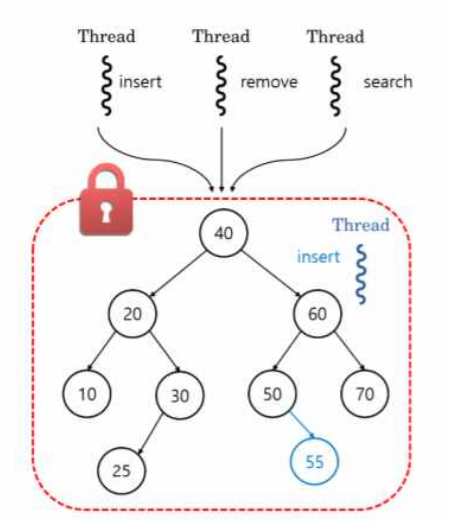
: 둘 이상의 입력 또는 조작이 동시에 이루어져, 프로그램의 결과가 입력, 조작의 순서에 의존하여 동작하게 되어버리는 상황을 Race Condition이라고 한다.
동시적 조작에 있어, 공유되는 부분을 Critical Section이라고 한다.

Binary Search Tree (이진탐색트리)

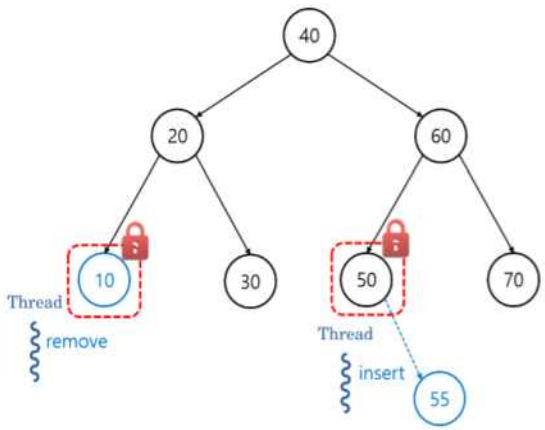
: 노드마다 key값을 가지며 left, right 두 개의 child link를 갖는 자료구조이며, 임의의 한 노드를 삽입, 삭제, 탐색하는 데 유리하다.
정의 - 모든 원소는 키를 가지며 동일한 키가 없다.
왼쪽 서브트리에 있는 key들은 루트의 key값보다 작다.
오른쪽 서브트리에 있는 key들은 루트의 key값보다 크다.
왼쪽, 오른쪽 서브트리도 이진탐색트리이다.

Coarse-grained Lock & Fine-grained Lock

: Coarse-grained Locking 방식은 Critical Section의 범위를 크게 잡아 많은 양의 데이터를 한 번에 보호하는 방식이다.
Fine-grained Locking 방식은 Critical Section의 범위를 세분화하여 각 부분마다 Lock을 사용하는 방식이다.



<BST with Coarse-grained lock>



<BST with Fine-grained lock>

Ⅱ. 프로젝트 시행

1. 실행 화면

```
os-lecture@os-lecture:~/2019_DKU_OS/lab2_sync$ ./lab2_bst -t 6 -c 1000000

=====  Multi thread single thread BST  insert experiment  =====

 Experiment info
    test node           : 1000000
    test threads        : 6
    execution time      : 2.813555 seconds


 BST inorder iteration result :
    total node count    : 1000000

=====  Multi thread coarse-grained BST  insert experiment  =====

 Experiment info
    test node           : 1000000
    test threads        : 6
    execution time      : 3.353203 seconds


 BST inorder iteration result :
    total node count    : 1000000

=====  Multi thread fine-grained BST    insert experiment  =====

 Experiment info
    test node           : 1000000
    test threads        : 6
    execution time      : 3.608314 seconds


 BST inorder iteration result :
    total node count    : 1000000

=====  Multi thread single thread BST  delete experiment  =====

 Experiment info
    test node           : 1000000
    test threads        : 6
    execution time      : 2.930029 seconds


 BST inorder iteration result :
    total node count    : 1000000

=====  Multi thread coarse-grained BST  delete experiment  =====

 Experiment info
    test node           : 1000000
    test threads        : 6
    execution time      : 4.230299 seconds


 BST inorder iteration result :
    total node count    : 1000000
세그멘테이션 오류 (core dumped)
os-lecture@os-lecture:~/2019_DKU_OS/lab2_sync$
```

Ⅲ. 프로젝트 결과

2. 박민혁 고찰

처음에 Binary Search Tree에 관해서 과제를 하는 것이기 때문에, 어렵지 않겠다고 생각했다. Lock 구현도, 이론적으로 잘 알고 있다고 생각했다. 하지만 생각 미스였다. 알고리즘 시간이나, 자료구조 시간에 Binary Search Tree 구현에 관한 과제를 했었다. 그 당시에도 구현에 어려움이 있었지만, 현재는 Binary Search Tree 구현에는 자신감이 있었다. 그래서 normal insert 와 normal remove는 쉽게 구현 할 수 있었다.

하지만, Lock 구현이 생각보다 쉽지 않았다. insert 함수에 coarse-grained lock insert는 몇 번 정도의 시행착오만 겪고 구현에 성공 하였다. 하지만 fine-grained lock insert 부분이 많이 막혔다. 처음에는 부모에만 Lock을 걸어 구현을 했었다. 하지만 계속 thread가 죽어서, 상민이와 고민을 했다. 그러다 헤더파일에 left child와 right child도 서로 mutex를 가지고 있어서, Lock을 가질 수 있었고, left child와 right child에도 같이 Lock을 걸어서 coarse-grained lock insert 와 fine-grained lock insert를 구현 할 수 있었다. 그러나 문제는, insert 함수에서 thread 수를 적게 하면, fine-grained lock 함수가 coarse-grained lock 보다 빠르게 나타 났는데, thread 수를 많게 하니, coarse-grained 함수가 더 빨리 수행 되었다.

그 다음으로, remove 함수가 수행 되던 도중에, thread가 계속 죽었다. insert 함수에서 coarse-grained lock도 별 시행착오 없이 진행 되었는데, remove에선 coarse-grained에서도 thread가 죽었었다. 그러던 도중, 상민이가 pthread_mutex (&p->mutex) 와 return LAB2_SUCCESS를 한 번에 묶어서 success로 묶어서 실행하니, coarse-grained remove 부분은 해결이 되었다. 쉽게 해결한 것 같지만, 여러 번의 시행착오를 겪었다. 하지만, 우리는 Binary Search Tree remove를 다른 학우들과 다르게, 구현 했는데, 그것이 문제가 되었다. 코딩이 너무 복잡해졌고, 우리가 주로 사용했던 코딩이 아니어서 오류를 잡기 힘들었다. 하지만 수정하지 않고, Binary Tree Search remove 부분을 상민이와 조금 특별하게 구현하고 싶어서, 수정을 하지 않았다. 그래서 계속 lock 거는 위치만 수정을 계속 했다. 하지만 역시나 오류가 계속 떴다. 뜨는 부분을 계속 로그를 걸어서 잡으려 했지만, 왜 죽는지 잘 못 찾았다. 상민이와 과제 하면서, 한 fine-grained remove 함수에서만 100번 넘게 오류가 뜬 것 같다. 저번 과제 scheduling과제는 오류를 금방 금방 잡을 수 있었다. 하지만, thread를 처음 다루는 코딩이어서 그런지 감이 잘 안 잡혔다. 오류가 나도, 어디서부터 잡아야 되는지 잘 몰랐고, fine-grained insert 함수와 계속 비교하면서, 그림도 그려 가면서, 코딩을 했지만 계속적으로 오류가 떴다. 선배한테도 물어 봤지만, 우리 코드가 자기가 사용 하던 Binary Search Tree 코딩이 아니어서 잘 모르겠

다고 했다. 그래서 상민이랑 둘이서만 과제 기간이 끝나기 전까지 둘이서 계속 고민을 했다. 아마 과제를 냈을 때는, 완성이 되 있거나 미완성 본으로 제출을 할 것이다.

discussion을 먼저 쓰는 이유는 과제 제출 기간이 거의 다 와 가는데 아직 완성을 못해서 우선 discussion을 먼저 쓰기로 했다. 그리고 과제를 닥쳐와서 한 것이 아닌, 시험이 끝나고 조금 쉬고 나서부터 과제를 시작 했는데도, 아직 미완성 상태이다. 이런 적은 처음이여서 조금 당황스럽기도 하지만, 그래도 열심히 하려고 노력은 했다. 비록 bonus 문제이지만, 해결 하고 싶다. 그래서 만약 미완성인 상태에서 제출을 하게 되면, 상민이와 구현을 꼭 하려고 한다. 과제를 하면서, 느낀 점은 시스템프로그래밍 수업 때 하던 과제는 과제도 아니구나. 라는 생각을 했고, 이제 정말 어려운 과제와 어려운 수업을 듣는 다는 것을 깨달았다. 그래도 과제에 시간을 이렇게 오래 들인 적은 처음이었고, 마치 개발자 체험을 하는 기분이었다.

1. 이상민 고찰

이번 과제는 이진탐색트리를 이용해 lock을 구현하는 과제였다. 기본적으로 트리를 구현하는 것은 어렵지 않았다. 작년 자료구조 수업 때부터 꾸준히 해왔고, 갖고 있는 자료도 많았다. 이진탐색트리의 연산으로는 중위 순회, 삽입, 삭제 이렇게 4가지를 구현했다.

중위 순회는 순환 호출 함수를 사용했다. 먼저 노드 값이 NULL값인지를 확인하고, NULL이 아닌 경우 왼쪽 자식 노드, 현재 노드, 오른쪽 자식 노드 순서로 순회했다.
삽입 함수는 이번 알고리즘 중간고사 시험 문제로도 나왔었다. 그래서 막힘없이 한 번에 코드를 짤 수 있었다. 삽입을 하기 위해서는 우선 탐색 과정이 필요하다. 트리의 root가 NULL인지를 확인하고 만약 NULL 즉, 비어있는 트리인 경우 매개변수로 받는 new_node를 그대로 넣어준다. 그렇지 않을 경우 이진탐색트리 정의에 의해 현재 key값보다 삽입하려는 노드의 key값이 더 작으면 왼쪽 자식 노드에, 그 반대의 경우에는 오른쪽 자식 노드에 삽입하고 LAB2_SUCCESS를 반환한다. 만약 삽입하려는 key값이 존재한다면 LAB2_ERROR를 반환한다. 이것을 이용해 비교적 간단해 보이는 coarse-grained lock을 먼저 구현해보았다. 처음에는 단순히 함수 처음과 끝에 lock과 unlock을 걸었는데 그랬더니 제대로 lock이 풀리지 않았다. 그래서 insert 함수 시작 부분에 lock을 걸고, 각 노드를 삽입한 후 unlock을 걸어 해결했다. fine-grained lock은 노드를 삽입하는 부분이 critical section이기 때문에 그 부분 앞뒤로 lock과 unlock을 걸었다.
자료구조 수업 시간에도 느낀 것이지만 삭제 함수는 기본적인 이론은 쉬운데 구현하기가 까다로웠다. 크게 보면 세 부분으로 나눌 수 있다. 삽입 함수와 같이 탐색 과정을 거치고, 삭제 노드를 대체할 노드를 탐색한 후 노드를 삭제하는 것이다. 우선 탐색 과정에서 errNum이라는 변수를 0으로 초기화해서 사용했다. 삭제하고자 하는 노드의 key값과 동일한 key값을 찾은 경우 errNum 변수를 1로 변화시키고, 찾지 못한 경우에는 LAB2_ERROR를 반환했다. 또한 case를 편하게 나누기 위해 childNum이라는 변수에 자식 노드의 수를 저장해서 사용하도록 구현했다. 자식 노드가 없는 경우에는 단순 삭제로 구현했다. 자식 노드가 1개인 경우에는 왼쪽 자식이 있는 경우와 오른쪽 자식이 있는 경우를 나누어서 삭제하도록 해주었다. 마지막으로 자식 노드가 2개인 경우에는 원래 두 가지 경우로 구현할 수가 있다. 왼쪽 서브트리의 가장 큰 key값을 가진 노드로 대체하거나 오른쪽 서브트리의 가장 작은 key값을 가진 노드로 대체하는 것이다. 우리는 전자를 선택했고, 왼쪽 서브트리에서 오른쪽 자식이 없을 때까지 반복문을 돌며 탐색했다. 삽입 함수와 마찬가지로 coarse-grained lock 먼저 구현해 보았다. 삭제 함수의 처음과 종료 전에 mutex라는 전역 변수를 이용하여 lock을 걸었다. 문제는 fine-grained lock이었다. 삭제하는 과정에서 이루어지는 모든 노드에 대해 lock을 관리해야 하는데 끝내 성공하지 못하고 계속 세그멘테이션 오

류가 떴다. 인터넷도 많이 찾아보고 했지만 해결하지 못했다. 어쩌면 해당 노드가 어떤 노드에 영향을 미치고, 어떠한 식으로 이동되고 변경되는지에 대한 이해가 부족해서 lock을 과하게 걸어준 탓이라고 생각한다.
tree_delete 함수에서는 tree_create 함수에서 동적 할당한 메모리를 free 함수를 이용해서 해제해주었다. 또한 node_delete 함수도 마찬가지로 node_create 함수에서 동적 할당한 메모리를 free 함수로 해제해주었다. 여기에서는 trylock을 사용했는데, trylock은 lock이 걸려있지 않으면 lock을 걸고 0을 반환해준다. 반면에 lock이 걸려 있는 경우에는 EBUSY라는 error를 반환해준다.

코드 자체가 이미 길어질 대로 길어졌고, 더이상의 수정이 어려울 것 같아 이대로 프로젝트를 제출하는 것이 상당히 아쉽다. 사실 시스템 프로그래밍 수업 때부터 이번 운영체제 수업까지 어떠한 방법을 써서라도 프로젝트를 항상 끝마쳤는데 처음으로 마무리를 못지었다. 코드가 전체적으로 꼬였고 그로 인해 발생한 중첩된 lock 사용으로 인해 lock 교착 상태에 빠진 것 같다. 이번 프로젝트처럼 길고 복잡한 코드를 구현할수록 더 최적화되고 깔끔하게 작성해야 이러한 문제점이 발생하지 않음을 깨달았다. 또 아쉬운 점은 insert_fg 함수의 구현 시간이 너무 빠르다는 것이다. lock을 걸어주기 때문에 그냥 insert 함수보다 수행 시간이 길어야 하는 것이 당연한데 왜 더 빠르게 측정되는지 모르겠다. 그런데 신기하게도 thread 개수를 더 늘려주면 우리가 원하는 수행 시간이 측정되었다. 다음 Lab3 프로젝트는 더 열심히 해서 깔끔하게 완성시키고 싶다.