CSE 437

REAL TIME SYSTEM ARCHITECTURES

Homework 1

A Thread-safe Set
Project Report

Muhammed Bedir ULUÇAY
1901042697

Content:

ABSTRACT

We are implementing a thread-safe set data structure. There are various way of implementing a set data structure. It can be use Tree set, Hash linked set, List set etc. and also there are various way of avoiding race conditions using some tools. Some of them are supported by operating system (OS) like conditions variables, semaphores, atomic operations etc to avoid race conditions and undefined behavior.

In this project I implemented a Thread Safe Lock-free Tree Set Data Structure. Using c++20 new features and gcc 12.1.0 version.

REQUIREMENTS

Add compile parameter -std=c++2a standard.
Use g++ 12.1.0 compiler (updated version current date).

INTRODUCTION

Set is a data structure that stores unique elements of the same type. In the implementation I use the tree structure. Because of the tree structure design the comparison is easy to other list approach If the value is smaller than the current value go left if the value is greater than go right. If the comparison is equal then the item already in the list.

So the class template argument must be comparable object. For the comparison operator< and operator== is added to the class. If the class not implement these operators Thread-safe set returning an assert fault. Telling the user you need to implement these operators.

```
24      ThreadSafeSet(){
25          static_assert(has_less_than<T>, "T must have operator<");
26          static_assert(has_equal_to<T>, "T must have operator==");
27      }
```

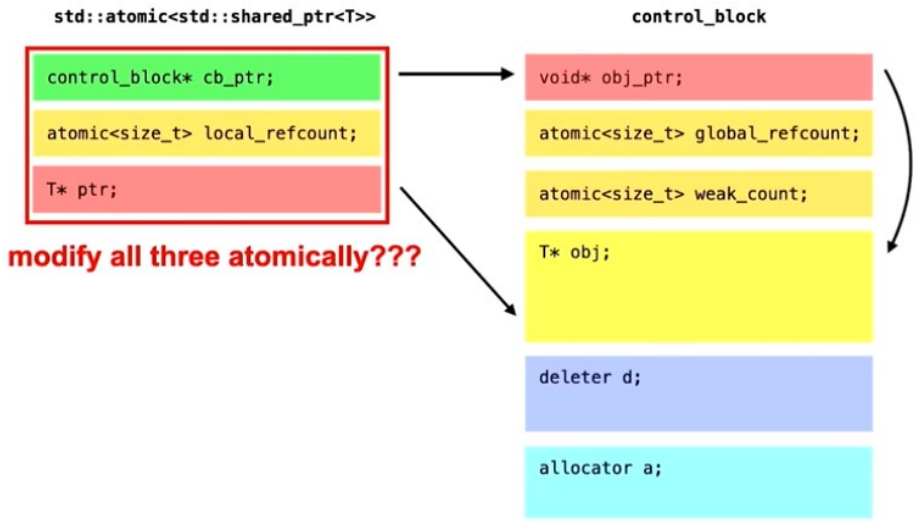
```
8   template<class L, class R = L>
9   concept has_less_than = requires(const L& lhs, const R& rhs)
10  {
11      {lhs < rhs} -> std::same_as<bool>;
12  };
13
14  template<class L, class R = L>
15  concept has_equal_to = requires(const L& lhs, const R& rhs)
16  {
17      {lhs == rhs} -> std::same_as<bool>;
18  };
```

If the object has these operators the class ready to store data.

The tree uses the shared pointers to go to the other nodes.

```cpp
struct Node
{
    T value;
    std::atomic<std::shared_ptr<Node>> left;
    std::atomic<std::shared_ptr<Node>> right;

    Node(const T& value) : value(value), left(nullptr), right(nullptr) {}
};
```

Until c++20 there is no direct exchange operation for the shared pointer. We are passing as an argument and we are hoping the exchange process will be atomic. Because of the shared pointer structure. The shared pointer has a lot of operation in the copying.



[A Lock-Free Atomic Shared Pointer in Modern Cpp - Timur Doumler - CppCon 2022]

For the lock-free approach a new features comes with the c++20 we can directly declare shared pointers in the atomic.

```cpp
bool compare_exchange_weak(std::shared_ptr<T>& expected, std::shared_ptr<T> desired)
{
    //c++11
    // return std::atomic_compare_exchange_weak(&ptr, &expected, desired);

    //c++20`
    return ptr.compare_exchange_weak(expected, desired);
}
```

There is open source and commercial implementation for the atomic shared pointer on the internet.
https://github.com/vtyulb/AtomicSharedPtr/blob/master/src/atomic_shared_ptr.h
https://github.com/facebook/folly/blob/main/folly/concurrency/AtomicSharedPtr.h
https://github.com/anthonywilliams/atomic_shared_ptr

These makes the operations are being atomic for the lock-free set.
For more detail watch the Timur Doumler - CppCon 2022 YouTube video please.

PROJECT DESIGN & IMPLEMENTATION

PART 1:

For the part 1 we have 1 writer and 1 reader. Even the operations are being an atomic. The operations a not executed at the same time. One before one after executing. So if the one writer is in the tree set actually there is not problem for the reader because it is not change the structure it just reading. The time reader is read if the object is exist it is exits return true if the object does not exist it is not exist return false. In that case no need protection for the reader. And there is 1 writer. Because of the just exist 1 writer also there is no need concurrency protection between writers. Because there is no other writer exist. It can iterate over as it wanted. Insertion and removing operations are making by its. The wrapper function just take an T argument and creating a local copy of the root node shared pointer. So we can iterate the other function. (Atomic objects not copyable in c++20) so we are sending a shared pointer to function.

```
std::atomic<std::shared_ptr<Node>> local;
local.store(root.load());

bool added = false;
while (true)
{
    std::shared_ptr<Node> v = local.load();
    std::shared_ptr<Node> result;

    COMPARE_EXCHANGE_WEAK(local, v, result);
    result = insert(v, value, added);

    if(result != nullptr){
        root.store(result);
        break;
    }
```

The function is started and proceeding as a normal tree set addition algorithm.

Insert:

```
std::shared_ptr<Node> insert(std::shared_ptr<Node> _local, T value, bool& added){

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    if(local.load() == nullptr){
        added = true;
        return std::make_shared<Node>(value);
    }else{

        if(value < local.load()->value){

            std::shared_ptr<Node> v = local.load()->left;
            COMPARE_EXCHANGE_WEAK(local.load()->left, v, insert(v, value, added));
        }
        if(!(value < local.load()->value) && !(value == local.load()->value)){
            std::shared_ptr<Node> v = local.load()->right;
            COMPARE_EXCHANGE_WEAK(local.load()->right, v, insert(v, value, added));
        }
    }

    return local.load();
}
```

The exchanged operations are made in a while loop. There is no need a protection for the critical region. Removing, searching and size operations are same as adding operations for the normal binary search tree algorithm.

Remove:

```
std::atomic<std::shared_ptr<Node>> local;
local.store(_local);

if(local.load() == nullptr){
    return local;
}
if(value < local.load()->value){

    std::shared_ptr<Node> v = local.load()->left;
    COMPARE_EXCHANGE_WEAK(local.load()->left, v, remove(v, value, removed));

}else if(!(value < local.load()->value) && !(value == local.load()->value)){

    std::shared_ptr<Node> v = local.load()->right;
    COMPARE_EXCHANGE_WEAK(local.load()->right, v, remove(v, value, removed));

}else{

    removed = true;
    if(local.load()->left.load() == nullptr && local.load()->right.load() == nullptr){

        std::shared_ptr<Node> v = local;
        COMPARE_EXCHANGE_WEAK(local, v, nullptr);

    }else if(local.load()->left.load() == nullptr){

        std::shared_ptr<Node> v = local;
        COMPARE_EXCHANGE_WEAK(local, v, local.load()->right);

    }else if(local.load()->right.load() == nullptr){
        std::shared_ptr<Node> v = local;
        COMPARE_EXCHANGE_WEAK(local, v, local.load()->left);

    }else{

        T max = findMax(local.load()->left);
        local.load()->value = max;

        std::shared_ptr<Node> v = local.load()->left;
        COMPARE_EXCHANGE_WEAK(local.load()->left, v, remove(v, max, removed));
        local.load()->right.store(remove(local.load()->right, local.load()->value, removed));
    }
}

return local;
```

Search:

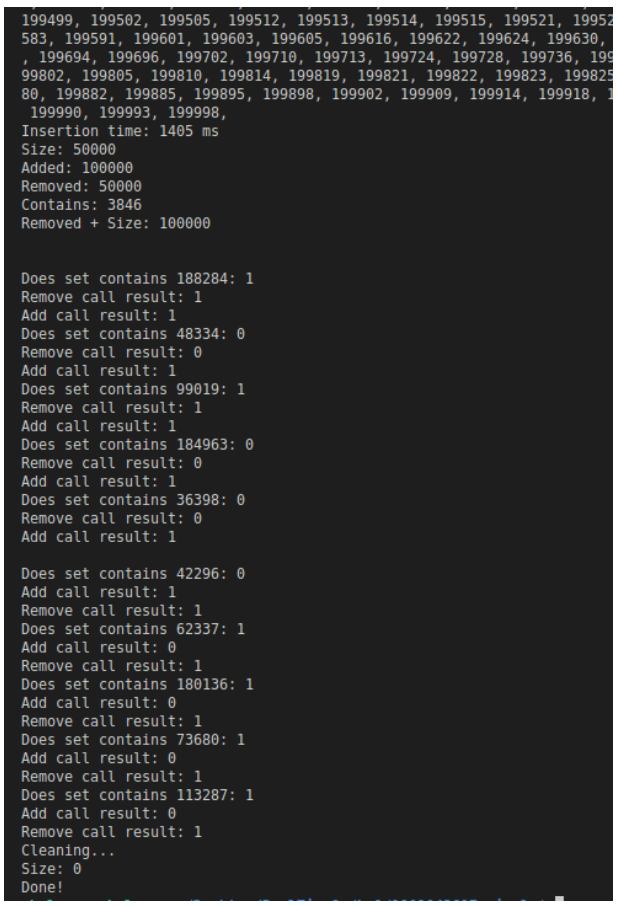
```
bool search(std::shared_ptr<Node> _local, T value) const {

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    if(local.load() == nullptr){
        return false;
    }
    if(value < local.load()->value){
        return search(local.load()->left, value);
    }else if(!(value < local.load()->value) && !(value == local.load()->value)){
        return search(local.load()->right, value);
    }else{
        return true;
    }
}
```

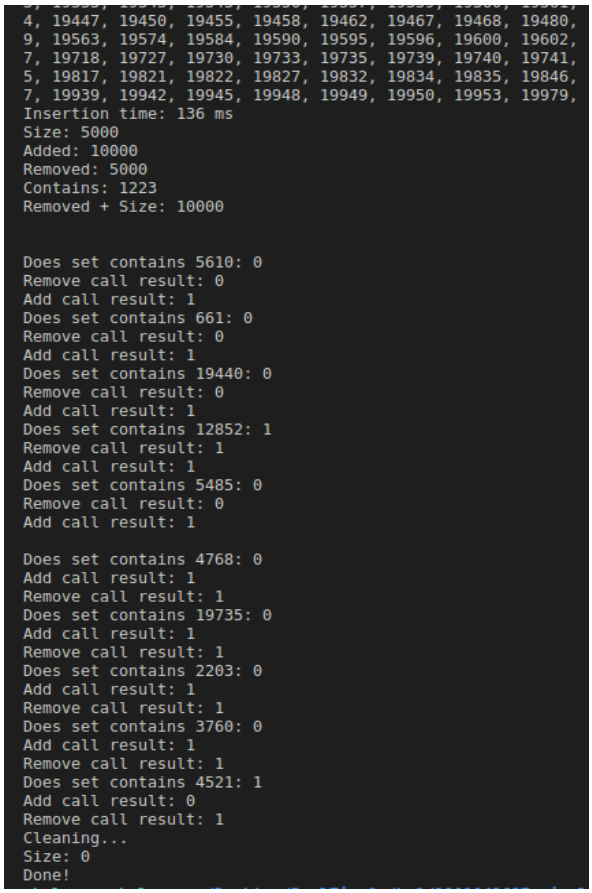
Size & Iterate:

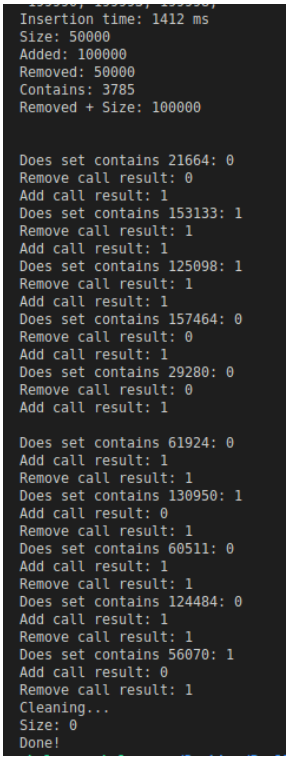
```
int size(std::shared_ptr<Node> local) const {
    if(local == nullptr)
        return 0;
    else
        return 1 + size(local->left) + size(local->right);
}


void iterate(std::shared_ptr<Node> local, const std::function<void(const T& obj)>& func) const {
    if(local != nullptr){
        iterate(local->left, func);
        func(local->value);
        iterate(local->right, func);
    }
}
```

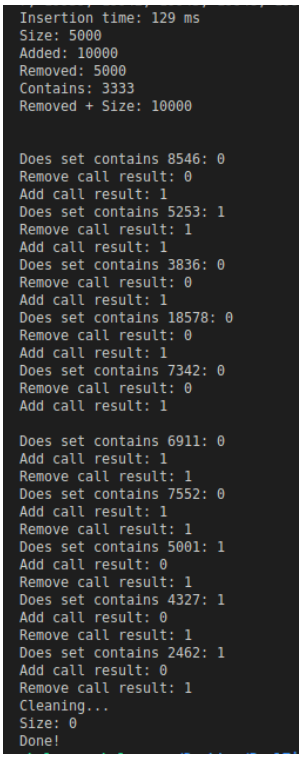
For the making test you can change the sleep settings to see much more different result.

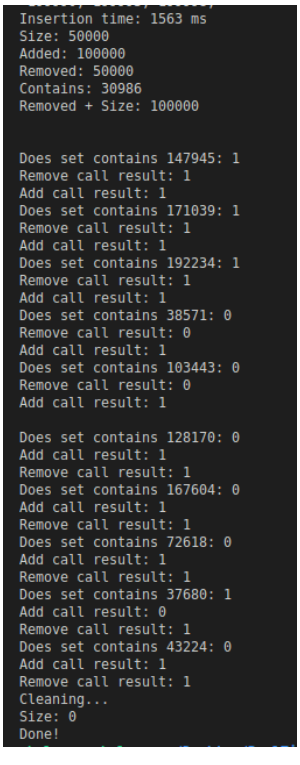
PART 1 TEST RESULT:











PART 2:

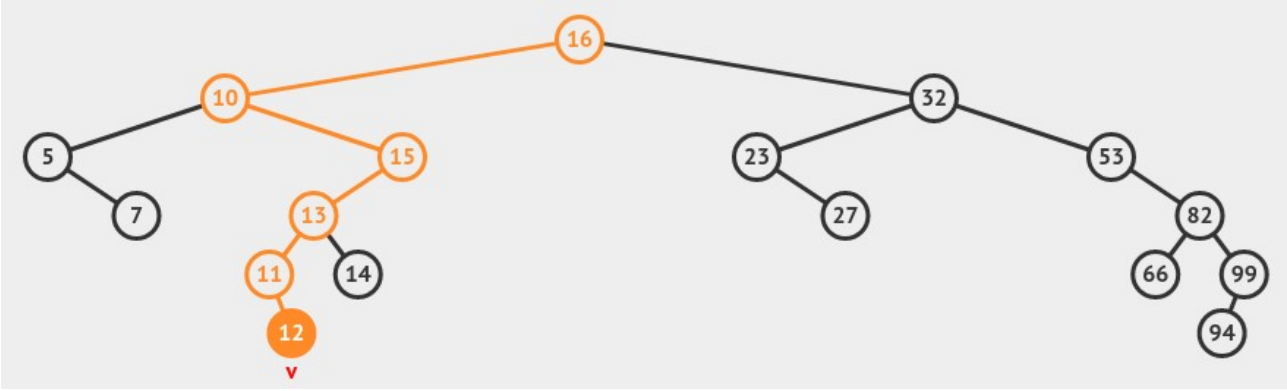
In this part we are going to implement thread-safe tree set for multiple reader and multiple writer. There are various ways for the implementation some of them are working perfect but take more time according to the other some of them are missing %3-%5 percent (Because of the tree structure there are a lot of unrelated branch not effecting each other) but it is %70 - %80 percent faster according to the other. Even if we don't put a lock mechanism It just missing %5-%7 percent if the the size of data is not important for you the more important thing is the fast you can check it out.

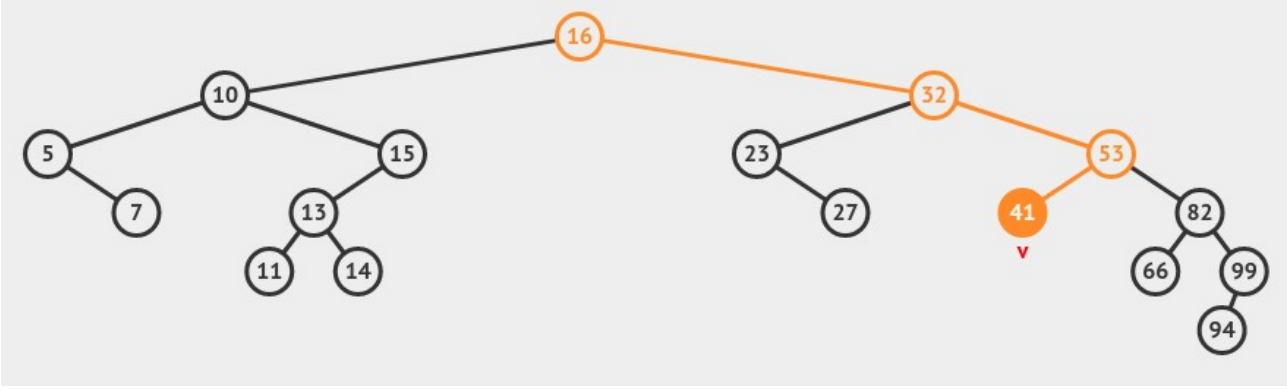
I will explain two different implementation first one is more complex according to the other one but it would has a much more speed according to the second implementation. I implement the second one because of the simplicity.

First Implementation:

There are a lot of list not depend on each other. for example one adding operation add new node left branch, other adding operation add new node right branch these are totally different rote. This allows us the multiple writer can access at the same time without waiting other. But how can we achieve this addition? For that each node need a marked boolean value if the node marked as doing an operation on it. The operation request waiting on a std::atomic_flag variable and wait until the atomic_flag is become true. But how we can make it true. As a result each node path start from the root variable. We need to make the flags true in some where. For this purpose we are sending the parent and grand parent address to the child. If the current check is on a node we are remark the grand parent node's flag. So the other node can pass the grand parent node.

For example:
We want to add 12, and 41 at the same time;



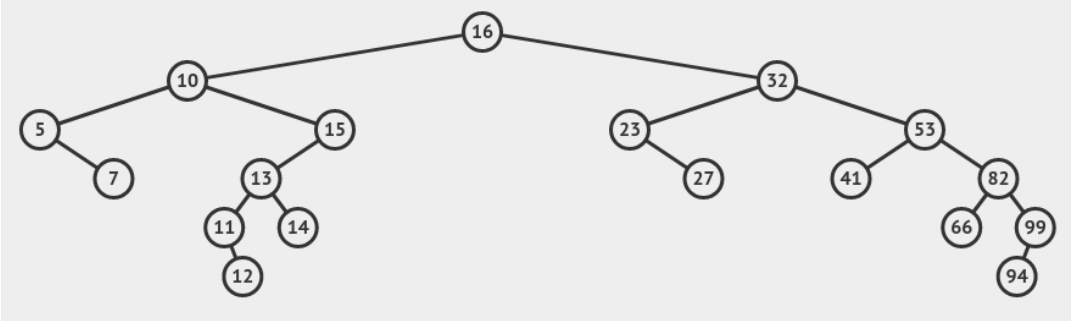


first 12 take the root and start to continue with std::atomic_flag locking.

Locked 16
Locked 10
Locked 15 and Unlock 16
Locked 13 and Unlock 10
Locked 11 and Unlock 15
Create new node 12
Assign left of 11
Come back with unlocking

41 start adding operations
Locked 16
Locked 32
Locked 53 and Unlock 16
Create new node 41
Assign left of 53
come back with unlocking

As a result:



Ps: I try to implement this method but time because of the time constraint I could not finish the implementation, but in the theory it seems work. I did not implement and test it.

```
std::shared_ptr<Node> add(std::shared_ptr<Node> _local, T value, std::shared_ptr<Node> parent, std::shared_ptr<Node> grand_parent,

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    // if(local.load() != nullptr)
    //     local.load()->mark();

    // if(grand_parent != nullptr && grand_parent->is_marked())
    //     grand_parent->un_mark();

    // if(local.load() != nullptr && local.load()->is_marked())
    //     local.load()->wait_unlock();


    if(local.load() == nullptr){
        added = true;
        return std::make_shared<Node>(value);
    }else{

        // local.load()->wait_left_unlock();
        if(value < local.load()->value){
            while(true){
                std::shared_ptr<Node> v = local.load()->left;
                if(local.load()->left.compare_exchange_weak(v, add(v, value, local.load(), parent, added))){
                    break;
                }
            }
            // local.load()->left = add(local.load()->left, value, local.load(), added);
        }
        // local.load()->un_mark_left();

        |     You, 11 hours ago • kajdladsaldjkasd …
        // local.load()->wait_right_unlock();
        if(!(value < local.load()->value) && !(value == local.load()->value)){
            while(true){
                std::shared_ptr<Node> v = local.load()->right;
                if(local.load()->right.compare_exchange_weak(v, add(v, value, local.load(), parent, added))){
                    break;
                }
            }
            // local.load()->right = add(local.load()->right, value, local.load(), added);
        }
        // local.load()->un_mark_right();


    }

    // if(local.load()->is_marked()){
    //     local.load()->un_mark();
    // }

    return local.load();
}
```

⚠ SonarLint is unable to analyze C and C++ file(s)
there is no configured compilation database.

Code in comment.

Second Implementation:

In the second implementation I used the std::atomic_flag variable for entrance to operate over the tree. Use of the atomic_flag mention that OS-2 lecture page 9.

```
atomic_flag disable_inputs = ATOMIC_FLAG_INIT;

void consumer() {
  MyData localBuffer;
  for (;;) {
    if (!atomic_test_and_set(&disable_inputs) {
```

This way just one request making an operation others are waiting for the finis of the current instruction. But it is still thread-safe, slow but thread-safe. Wrapper method entrance make the flag true and others are wait in a while loop. If the loop number bigger than a specific number the thread calling std::thread_yield() method. To give the its order to some other thread.

```
#define ATOMIC_FLAG_LOCK(lock) \
    int c = 0;\
    while(lock.test_and_set(std::memory_order_acquire)){\
        if(c++ >= 58){\
            std::this_thread::yield();\
        }\
    }\

#define ATOMIC_FLAG_UNLOCK(lock) \
    lock.clear(std::memory_order_release);\
```

We are making an lock in wrapper class and sending copy of the root to make an iteration over the tree.

```
ATOMIC_FLAG_LOCK(flag);
std::atomic<std::shared_ptr<Node>> local;
local.store(root.load());
bool added = false;
while (true)
{
    std::shared_ptr<Node> v = local.load();
    std::shared_ptr<Node> result;
    COMPARE_EXCHANGE_WEAK(local, v, result);
    result = add(v, value, added);

    if(result != nullptr){
        root.store(result);
        break;
    }
}
ATOMIC_FLAG_UNLOCK(flag);
```

Insertion:

```
// std::shared_ptr<Node> add(std::shared_ptr<Node> _local, T value, std::shared_ptr<Node> parent,
std::shared_ptr<Node> add(std::shared_ptr<Node> _local, T value, bool& added){

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    // if(gParent != nullptr){
    //     ATOMIC_FLAG_UNLOCK(gParent->marked);
    // }
    // if(local.load() != nullptr){
    //     ATOMIC_FLAG_LOCK(local.load()->marked);
    // }

    if(local.load() == nullptr){
        added = true;
        return std::make_shared<Node>(value);
    }else{
        if(value < local.load()->value){      You, 22 hours ago • asdadsljasd …

            std::shared_ptr<Node> v = local.load()->left;
            COMPARE_EXCHANGE_WEAK(local.load()->left, v, add(v, value, added));
            // local.load()->left = add(local.load()->left, value, local.load(), added);
        }

        if(!(value < local.load()->value) && !(value == local.load()->value)){
            std::shared_ptr<Node> v = local.load()->right;
            COMPARE_EXCHANGE_WEAK(local.load()->right, v, add(v, value, added));
            // local.load()->right = add(local.load()->right, value, local.load(), added);
        }
    }

    // ATOMIC_FLAG_UNLOCK(local.load()->marked);
    return local.load();
}
```

Remove:

```
std::shared_ptr<Node> remove(std::shared_ptr<Node> _local, T value, bool& removed){

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    if(local.load() == nullptr){

        return local;
    }
    if(value < local.load()->value){

        std::shared_ptr<Node> v = local.load()->left;
        COMPARE_EXCHANGE_WEAK(local.load()->left, v, remove(v, value, removed));

    }else if(!(value < local.load()->value) && !(value == local.load()->value)){

        std::shared_ptr<Node> v = local.load()->right;
        COMPARE_EXCHANGE_WEAK(local.load()->right, v, remove(v, value, removed));

    }else{

        removed = true;                (std::nullptr_t)nullptr
        if(local.load()->left.load() == nullptr && local.load()->right.load() == nullptr){

            std::shared_ptr<Node> v = local;
            COMPARE_EXCHANGE_WEAK(local, v, nullptr);

        }else if(local.load()->left.load() == nullptr){

            std::shared_ptr<Node> v = local;
            COMPARE_EXCHANGE_WEAK(local, v, local.load()->right);

        }else if(local.load()->right.load() == nullptr){

            std::shared_ptr<Node> v = local;
            COMPARE_EXCHANGE_WEAK(local, v, local.load()->left);

        }else{

            T max = findMax(local.load()->left);
            local.load()->value = max;

            std::shared_ptr<Node> v = local.load()->left;
            COMPARE_EXCHANGE_WEAK(local.load()->left, v, remove(v, max, removed));

            local.load()->right.store(remove(local.load()->right, local.load()->value, removed));
        }
    }

    return local;
}
```

Search:

```
bool search(std::shared_ptr<Node> _local, T value) const {

    std::atomic<std::shared_ptr<Node>> local;
    local.store(_local);

    if(local.load() == nullptr){
        return false;
    }
    if(value < local.load()->value){
        return search(local.load()->left, value);
    }else if(!(value < local.load()->value) && !(value == local.load()->value)){
        return search(local.load()->right, value);
    }else{
        return true;
    }
}
```

Size & Iterator:

```
int size(std::shared_ptr<Node> local) const {
    if(local == nullptr)
        return 0;
    else
        return 1 + size(local->left) + size(local->right);
}


void iterate(std::shared_ptr<Node> local, std::function<void(T)> f) const {
    if(local != nullptr){
        iterate(local->left, f);
        f(local->value);
        iterate(local->right, f);
    }
}
```

I used local copy for the search, size and iterator functions to make it independence.
If the node exist do operation if not den return. This make reader function more flexible.

PART 2 TEST RESULT:

```
57, 99758, 99759, 99761, 99763, 99765, 99769, 99771, 99773, 99776, 99778
40, 99843, 99845, 99847, 99853, 99854, 99855, 99859, 99861, 99862, 99865
21, 99926, 99928, 99929, 99931, 99932, 99933, 99936, 99938, 99939, 99946

Insertion time: 1070 ms
Size: 33561
Added: 50000
Removed: 16439
Contains: 950
Removed + Size: 50000

Does set contains 66398: 1
Remove call result: 1
Add call result: 1
Does set contains 67537: 0
Remove call result: 0
Add call result: 1
Does set contains 29153: 0
Remove call result: 0
Add call result: 1
Does set contains 52641: 0
Remove call result: 0
Add call result: 1
Does set contains 54230: 0
Remove call result: 0
Add call result: 1

Does set contains 62910: 0
Add call result: 1
Remove call result: 1
Does set contains 95927: 1
Add call result: 0
Remove call result: 1
Does set contains 75935: 1
Add call result: 0
Remove call result: 1
Does set contains 24521: 0
Add call result: 1
Remove call result: 1
Does set contains 57631: 1
Add call result: 0
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/RealTimeOs/hw1/1901042697_multi$
```

```
5, 199776, 199777, 199779, 199780, 199784, 199788, 199789, 199790, 19979
199834, 199838, 199841, 199842, 199848, 199851, 199852, 199855, 199859,
910, 199913, 199914, 199918, 199919, 199922, 199923, 199924, 199926, 199
, 199977, 199978, 199980, 199989, 199990, 199992, 199993, 199996, 199998

Insertion time: 2266 ms
Size: 76671
Added: 100000
Removed: 23329
Contains: 392
Removed + Size: 100000

Does set contains 111752: 1
Remove call result: 1
Add call result: 1
Does set contains 195886: 1
Remove call result: 1
Add call result: 1
Does set contains 145297: 0
Remove call result: 0
Add call result: 1
Does set contains 159352: 0
Remove call result: 0
Add call result: 1
Does set contains 77697: 0
Remove call result: 0
Add call result: 1

Does set contains 94032: 0
Add call result: 1
Remove call result: 1
Does set contains 121071: 1
Add call result: 0
Remove call result: 1
Does set contains 46054: 1
Add call result: 0
Remove call result: 1
Does set contains 149264: 1
Add call result: 0
Remove call result: 1
Does set contains 152450: 0
Add call result: 1
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/RealTimeOs/hw1/1901042697_multi$
```

```
999807, 1999812, 1999814, 1999818, 1999819, 1999822, 1999823, 1999827, 19
0, 1999881, 1999882, 1999884, 1999885, 1999888, 1999889, 1999891, 199989
99932, 1999935, 1999936, 1999937, 1999939, 1999943, 1999945, 1999946, 19
, 1999990, 1999992, 1999999,

Insertion time: 32978 ms
Size: 729773
Added: 999997
Removed: 270227
Contains: 509
Removed + Size: 1000000

Does set contains 753832: 0
Remove call result: 0
Add call result: 1
Does set contains 1997087: 0
Remove call result: 0
Add call result: 1
Does set contains 538969: 0
Remove call result: 0
Add call result: 1
Does set contains 1595592: 1
Remove call result: 1
Add call result: 1
Does set contains 33645: 1
Remove call result: 1
Add call result: 1

Does set contains 377622: 0
Add call result: 1
Remove call result: 1
Does set contains 433461: 1
Add call result: 0
Remove call result: 1
Does set contains 1400900: 0
Add call result: 1
Remove call result: 1
Does set contains 1839316: 0
Add call result: 1
Remove call result: 1
Does set contains 692508: 0
Add call result: 1
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/RealTimeOs/hw1/1901042697 multi$
```

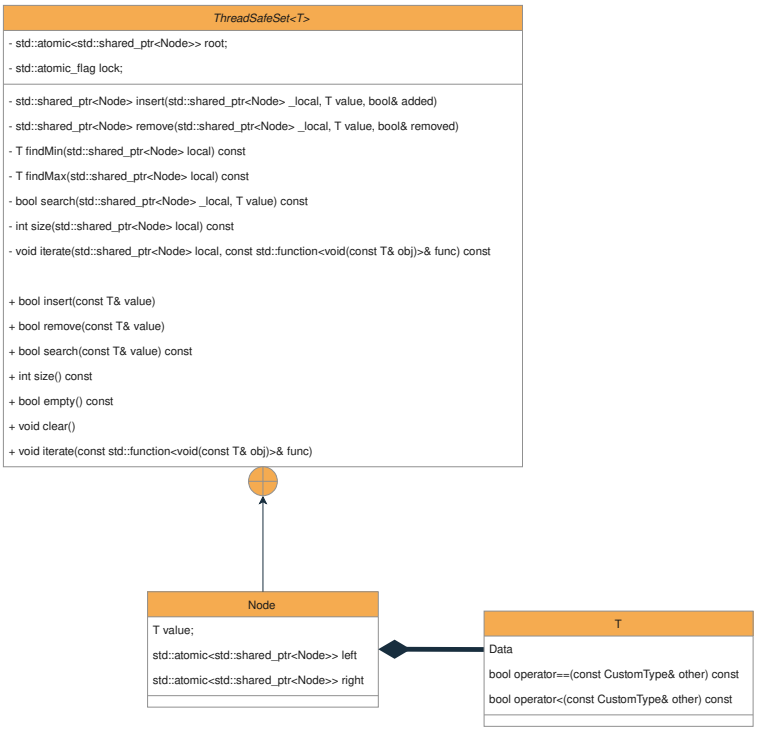
```
99773, 999779, 999780, 999781, 999782, 999784, 999791, 999794, 999800, 99
31, 999832, 999836, 999837, 999839, 999840, 999842, 999843, 999849, 99985
 999885, 999888, 999889, 999891, 999893, 999895, 999896, 999897, 999899,
9941, 999942, 999946, 999952, 999957, 999959, 999963, 999966, 999970, 999

Insertion time: 15043 ms
Size: 373869
Added: 500000
Removed: 126131
Contains: 688
Removed + Size: 500000

Does set contains 696252: 0
Remove call result: 0
Add call result: 1
Does set contains 720004: 0
Remove call result: 0
Add call result: 1
Does set contains 274406: 0
Remove call result: 0
Add call result: 1
Does set contains 56425: 0
Remove call result: 0
Add call result: 1
Does set contains 41551: 0
Remove call result: 0
Add call result: 1

Does set contains 929806: 1
Add call result: 0
Remove call result: 1
Does set contains 383903: 1
Add call result: 0
Remove call result: 1
Does set contains 90282: 1
Add call result: 0
Remove call result: 1
Does set contains 350363: 1
Add call result: 0
Remove call result: 1
Does set contains 83936: 0
Add call result: 1
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/RealTimeOs/hw1/1901042697 multi$
```

```
Insertion time: 2545 ms
Size: 90331
Added: 100000
Removed: 9669
Contains: 348
Removed + Size: 100000

Does set contains 154646: 0
Remove call result: 0
Add call result: 1
Does set contains 42240: 0
Remove call result: 0
Add call result: 1
Does set contains 75639: 1
Remove call result: 1
Add call result: 1
Does set contains 182775: 1
Remove call result: 1
Add call result: 1
Does set contains 4020: 0
Remove call result: 0
Add call result: 1

Does set contains 53147: 0
Add call result: 1
Remove call result: 1
Does set contains 92841: 1
Add call result: 0
Remove call result: 1
Does set contains 94087: 0
Add call result: 1
Remove call result: 1
Does set contains 90471: 1
Add call result: 0
Remove call result: 1
Does set contains 84732: 0
Add call result: 1
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/Re
```

```
Insertion time: 2662 ms
Size: 60190
Added: 100000
Removed: 39810
Contains: 1415
Removed + Size: 100000

Does set contains 159738: 1
Remove call result: 1
Add call result: 1
Does set contains 161052: 1
Remove call result: 1
Add call result: 1
Does set contains 193153: 0
Remove call result: 0
Add call result: 1
Does set contains 149514: 0
Remove call result: 0
Add call result: 1
Does set contains 190451: 0
Remove call result: 0
Add call result: 1

Does set contains 39038: 0
Add call result: 1
Remove call result: 1
Does set contains 90581: 0
Add call result: 1
Remove call result: 1
Does set contains 55346: 0
Add call result: 1
Remove call result: 1
Does set contains 137670: 0
Add call result: 1
Remove call result: 1
Does set contains 102565: 0
Add call result: 1
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/Re
```

```
Insertion time: 2493 ms
Size: 78514
Added: 100000
Removed: 21486
Contains: 926
Removed + Size: 100000

Does set contains 77938: 1
Remove call result: 1
Add call result: 1
Does set contains 150672: 1
Remove call result: 1
Add call result: 1
Does set contains 78172: 0
Remove call result: 0
Add call result: 1
Does set contains 127152: 0
Remove call result: 0
Add call result: 1
Does set contains 14541: 0
Remove call result: 0
Add call result: 1

Does set contains 176471: 1
Add call result: 0
Remove call result: 1
Does set contains 86342: 1
Add call result: 0
Remove call result: 1
Does set contains 107659: 1
Add call result: 0
Remove call result: 1
Does set contains 156444: 0
Add call result: 1
Remove call result: 1
Does set contains 61933: 1
Add call result: 0
Remove call result: 1
Cleaning...
Size: 0
Done!
mbulucay@mbulucay:~/Desktop/Re
```

CLASS DIAGRAMS:



Bonus Implementation:

I try to implement classic classic reader writer implementation with atomic operations in c++20. But it did not worked at some point it entrance the deadlock.

```cpp
std::atomic_flag read_flag = ATOMIC_FLAG_INIT;
std::atomic_flag write_flag = ATOMIC_FLAG_INIT;

std::atomic_flag lock_flag = ATOMIC_FLAG_INIT;
std::atomic<bool> lock_ = false;

std::atomic<int> active_read_count = 0;
std::atomic<int> active_write_count = 0;
std::atomic<int> waiting_read_count = 0;
std::atomic<int> waiting_write_count = 0;
```

```cpp
// WRITER ENTRY LOCK
lock();
while((active_write_count + active_read_count) > 0){
    waiting_write_count++;
    unlock();
    do{
        write_flag.wait(false);
    }while(active_read_count || active_write_count);
    lock();
    waiting_write_count--;
}
active_write_count = 1;
unlock();



wait_add_lock();
std::atomic<std::shared_ptr<Node>> local;
local.store(root.load());
bool added = false;
while (true)
{
    std::shared_ptr<Node> v = local.load();
    std::shared_ptr<Node> result;
    while(!local.compare_exchange_weak(v, result)){
        // Wait for compare_exchange_weak to succeed
    }
    result = add(v, value, nullptr, nullptr, added);

    if(result != nullptr){
        root.store(result);
        break;
    }
}
add_unlock();

// WRITER EXIT LOCK
lock();
active_write_count = 0;
if(waiting_write_count > 0){
    write_flag.notify_one();
}else if(waiting_read_count > 0){
    read_flag.notify_all();
}
unlock();
```

```cpp
// std::atomic<bool> lock;
void lock(){
    while(lock_.exchange(true)){
        // Wait until lock_ assign false
    }
}

void unlock(){
    lock_.store(false);
}
```