CS4204 Report
170025298 04/03/2021

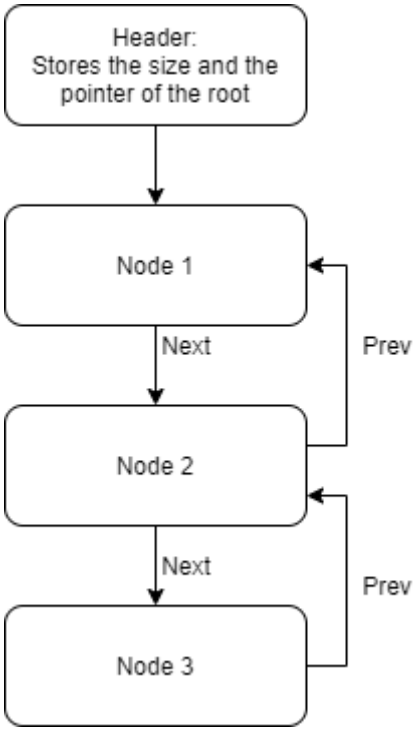
Section 1: Introduction

Concurrency happens when we try to use multiple threads for handling one or more practical issues. At the same time, List is being considered as a generally used datatype for handling data since its internal functions make it available for more comfortable use. However, in most conditions, multiple threads read and write the same List tend to cause the data race condition, which may lead to segment fault, and the programme would crash due to the data race. It is reasonable to implement a list that satisfies reading and writing by multiple threads for a more straightforward handling of data.

Section 2: Design

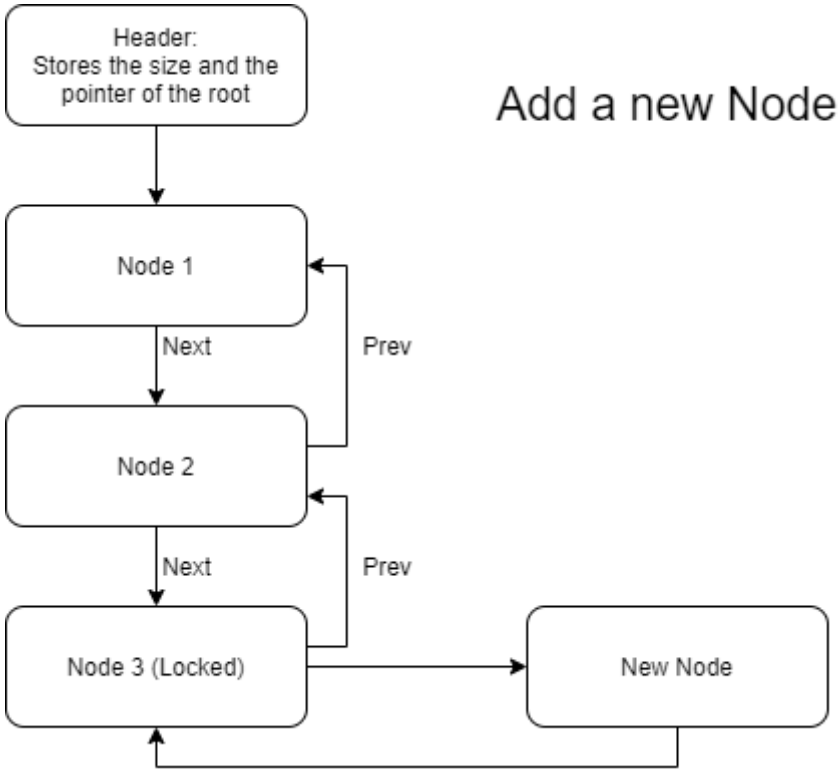
Section 2.1 Blocked Set

A blocked set is being considered as a traditional double linked list that supports working concurrently. The locker is to be stored in each node, whereas adding, removing, or containing will lock the single node and reject the other's visiting. The locker will only lock the node, and operations on the other part of the node will not be affected.



The figure showing above represents the structure of the blocked list. As is shown above, the blocked list consists of a header and a series of nodes. The detail of the list will be described below.

Section 2.1.1: Add a new node

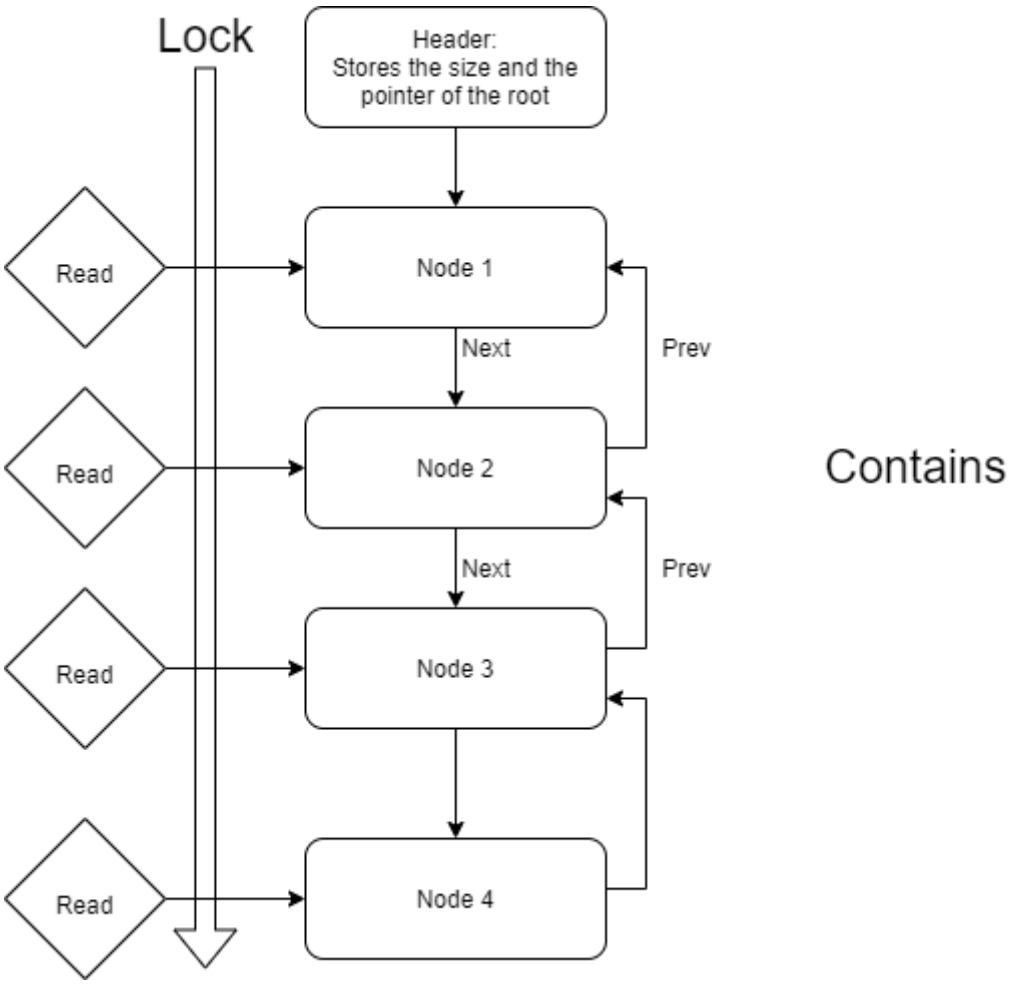


The figure showing above represents the process of adding a new node to the blocked list. As we can find that the node can store the data concurrently, but reading the pointer for the previous node may cause the data race (another thread may writing to the previous node either). The way of solving the issue is to lock the previous node to make sure that the new node just follows the previous node. The lock prevents the data race.

A special case for adding a new node happens when we would like to add the first node to the header, the locker will lock the header of nodes. In this case, the header will be locked, and the new node will be regarded as the root of the whole linked list.

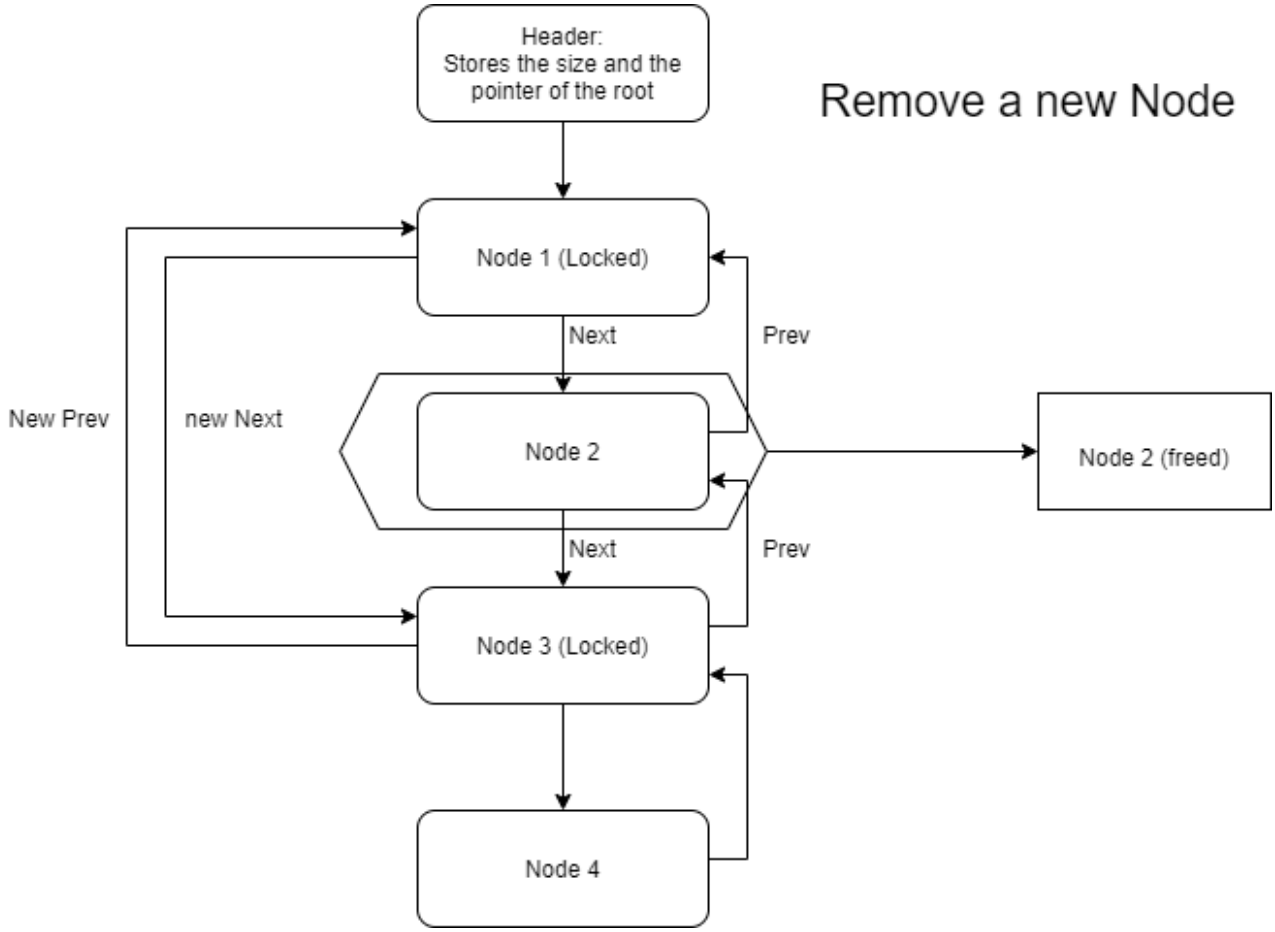
Note that two elements in the structure: previous and next, will be initialized with the NULL pointer in order that easier justifies whether there exists a node to follow. The root object will not store the address for the header in case that header is different from the node structure.

Section 2.1.2: Check whether the set contains the data



As the figure showing above, read the data of a node will lock the node first, once the data is found to be equal to the target, the function will return 1 (which represents true - C does not contains true without importing the Boolean package). While the pointer read through the whole set and unable to find the target data, the function will return 0 (represents False).

Section 2.1.3: Remove the node



As the figure showing above, removing the node should unbind the node from other nodes first. In this case, the previous pointer for the next node will points the previous node, and the next pointer of the previous node will points to the previous pointer of the next node. For preventing the data race, the previous node and the next node of the current node should be locked until they are connected.

When the node that we would like to delete is definitely retrieved, we would free the node and the node is now removed.

Section 2.1.4: Conclusion

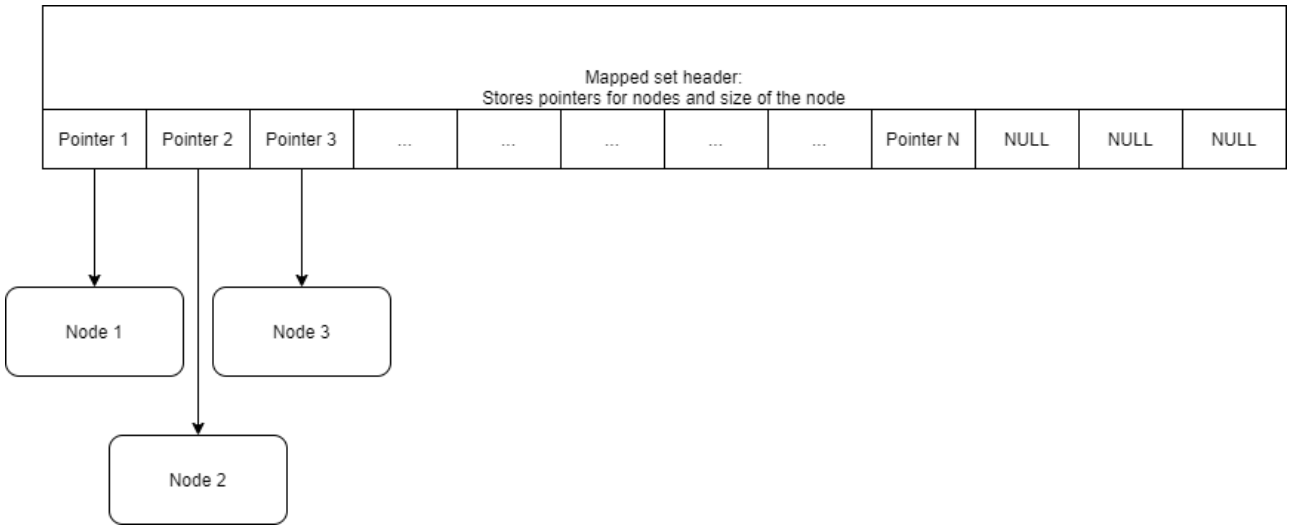
The blocked node represents the medium efficiency on retrieving and storing data. This happens because nodes are to be stored one by one, and the doubly linked list requires writing next and previous address for a more stable link between nodes. A improvement for this set may modifies the double linked list to just a single linked list in order that decrement times of writing.

Section 2.2 Mapped Set

A mapped set is being considered to be a higher efficiency set for handling data and preventing data race. The basic design is based on the consideration of the following:

- Operation on an array tends to be more efficient than operating on a linked list.
- The linked list is considered to be a set of pointers during the implementation.
- Locker should lock variables as less as possible.

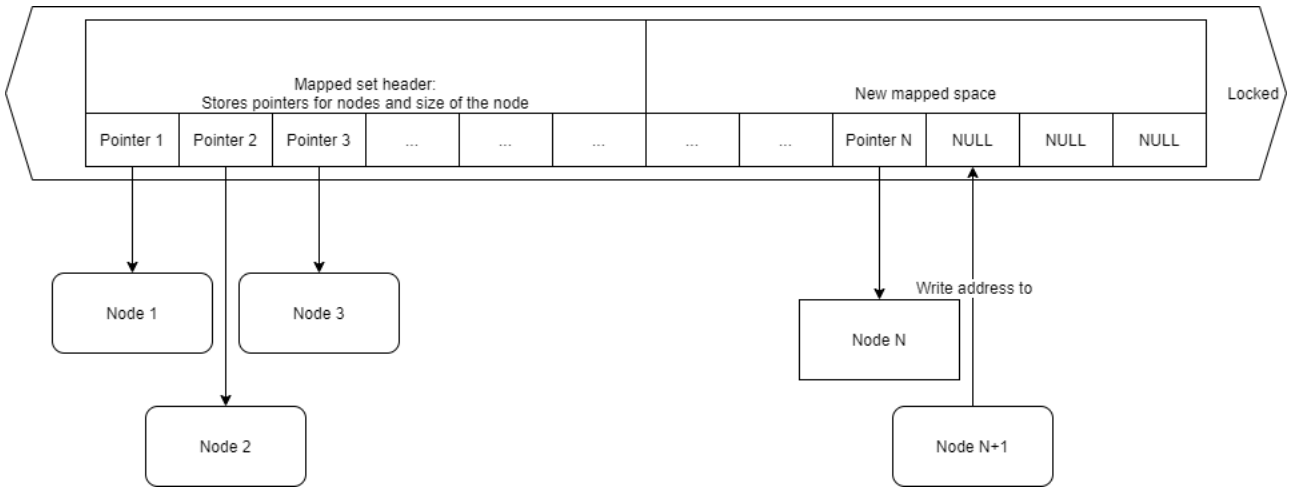
Hence, we implemented a mapped set (I would prefer to call it a hash set, even the set is not based on hash), mapping the data block's pointers to an array. Since we do only need to lock pointers, the set should operate faster than the blocked set showing above.



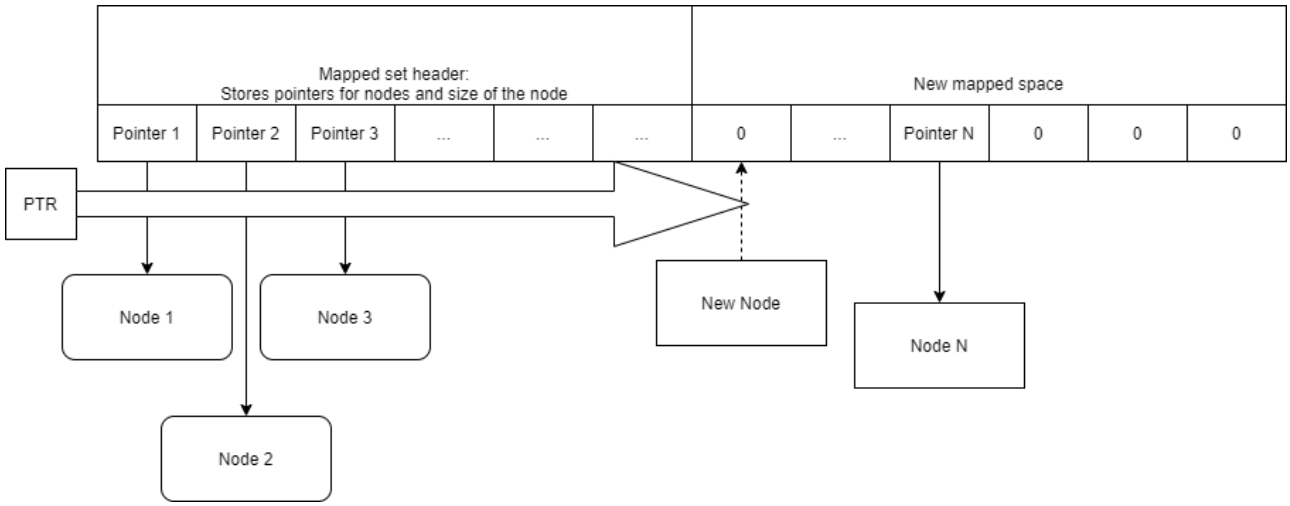
As is showing above, a mapped set is being considered as a whole map which maps all pointers that points to different nodes. Since we do not know how large would be the mapped set, the map should be extendable.

Section 2.2.1: Add a new node

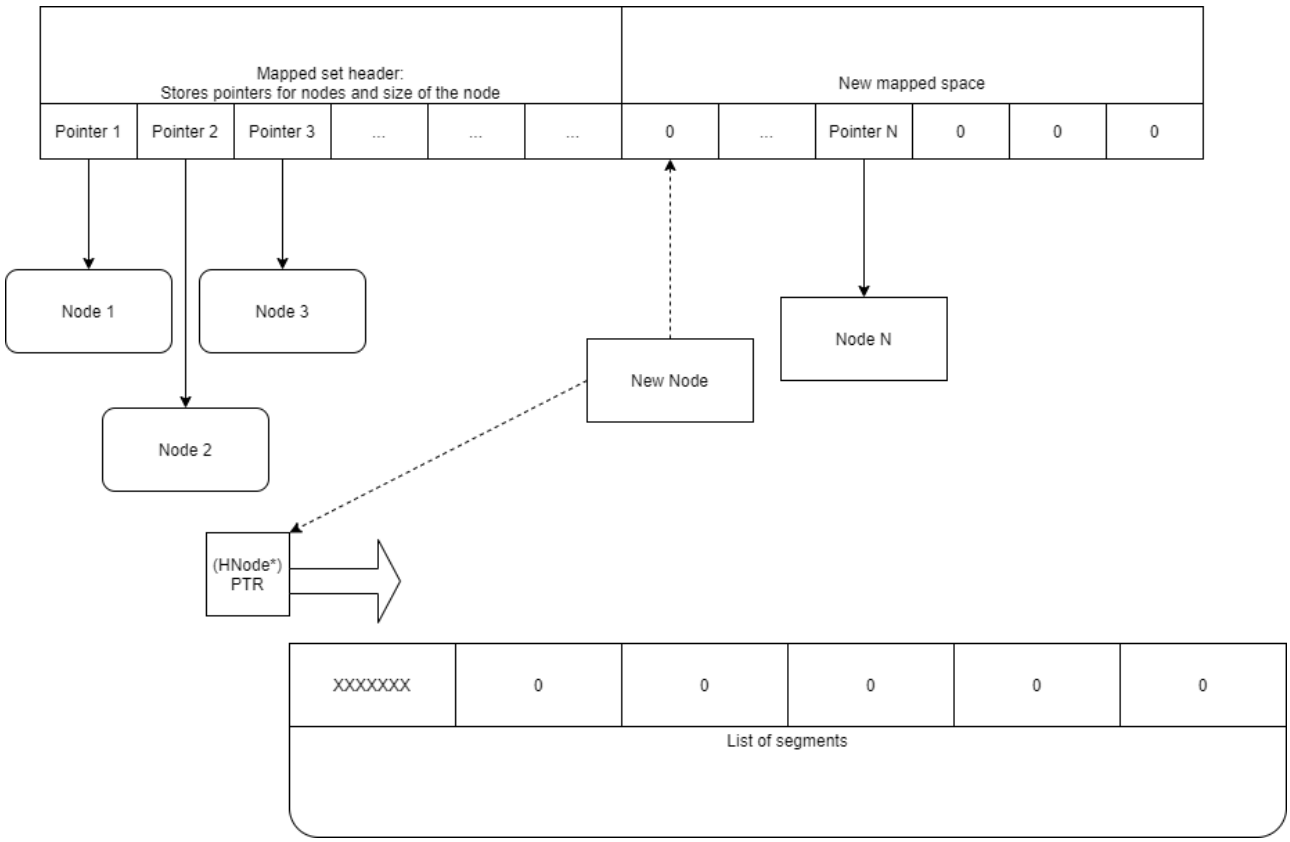
There are two conditions that we may need to consider. The first condition is that there is no removal function applies to the map. Adding a new node to the map is being considered as is shown as follows:



Nodes are directly add to the end of the list. This process runs quite fast as nodes are at the specific position of the map. However, when some nodes are deleted, we would need a way of handling these deleted nodes in order that prevents from segments. Hence we initialize the map with calloc() to initialize all pointers of the address array to 0, and we could find 0s in the map. The process is shown as follows:

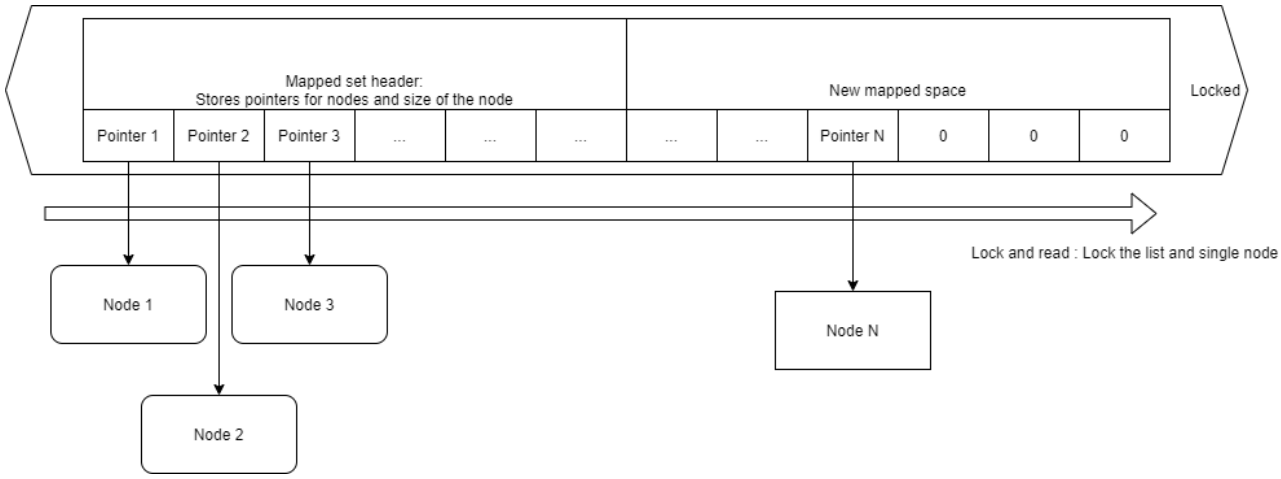


However, time efficiency is affected by the search process. For each searching for segments, the pointer have to look through the whole pointer array for checking whether it represents 0. An enhancement method for handling the segments may provides an array for handling these removal segments. The size of the array based on how many threads access to the map, and requires approximately size of (x + 5) for handling segments.



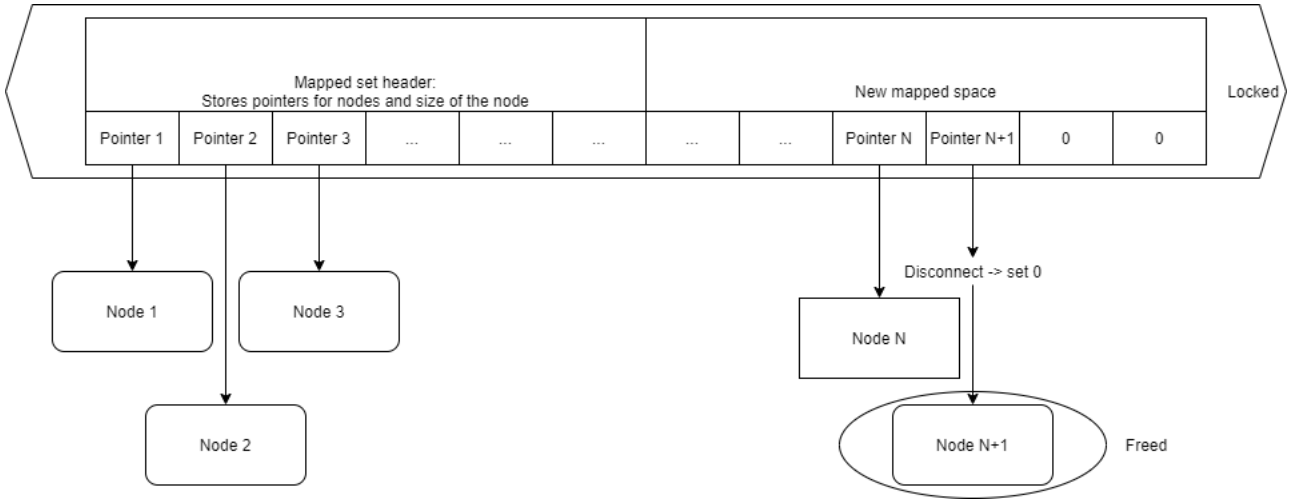
And we do only need to lock the map for increment the size of the map, which makes it faster for processing adding.

Section 2.2.2: Check whether the node exists



Contain runs as is shown above, checking whether the list contains the value look through the whole set. The function reads nodes according to its non-zero addresses and check whether there exists a node contains the value that equal to the expected value.

Section 2.2.3: Remove a node



Removing a node can be simple, we do only need to know where the node is and set the address of the map to 0, and free the unbind node directly. The locker will lock the whole set when looking through values of the set, but will not lock the process of checking whether the node contains the value.

Section 2.2.4: Conclusion

This mapped set represents higher efficiency the the blocked set, as the map does only retrieve data from pointers and run continuously. The map is unlocked in a short period when pointer access to the node, and other threads are able to visit the node.

Section 3: Implementation

The program implements the following features:

- A Blocked Set that locks the single node of the set - medium efficiency
    - Contains function
    - Add function
    - Remove function
- A Mapped Set that stores the data outside of the set, and stores address for the pointer in the pointer array - medium high efficiency
    - Contains function
    - Add function
    - Remove function

Section 4: Testing

For testing the efficiency of sets, we would test using random method for storing data into the set. We would test the efficiency of sets with 60, 300, 900 data and testing the time cost for each functions.

Section 4.1 Blocked Set

Small

```
/mnt/e/BILI/P2Y2S1/CS4204-Practical/Practical-1/cmake-build-debug/Practical_1
Time cost for current set is 163
Adding object 10000 to the list...
Checking containing the test number: 10000... status 1
Checking containing the test number: 2000... status 0
Checking the validation of object 10000 removal: 0
Checking the validation of object 2000 removal: 404
The process costs 32 for removing and containing test

Process finished with exit code 0
```

As is shown above, testing blocked set with 60 inputs costs 163ms, and costs 32ms in total for removing and searching.

Medium

```
ime cost for current set is 378
dding object 10000 to the list...
hecking containing the test number: 10000... status 1
hecking containing the test number: 2000... status 0
hecking the validation of object 10000 removal: 0
hecking the validation of object 2000 removal: 404
he process costs 111 for removing and containing test

rocess finished with exit code 0
```

Testing blocked set with 300 inputs costs obviously more time than that of 60 inputs. The medium amount of input costs 378ms for adding data, and 111ms for searching and removing.

Large

```
/mnt/e/BILI/P2Y2S1/CS4204-Practical/Practical-1/cmake-build-debug/Practical_1
Time cost for current set is 11438
Adding object 10000 to the list...
Checking containing the test number: 10000... status 1
Checking containing the test number: 2000... status 0
Checking the validation of object 10000 removal: 0
Checking the validation of object 2000 removal: 404
The process costs 238 for removing and containing test

Process finished with exit code 0
```

Similarly, large data input costs more that that of input the medium size data. The large input costs 11438ms for adding new node to data, and costs 238ms for searching and removing.

Section 4.2 Mapped Set

Small

```
Time cost for current set is 152
Invalid check: 2000: 0
Contain check: 1000: 1
Node removed for 1000.
Contain check: 1000: 0
The process costs 45 for removing and containing test
Process finished with exit code 0
```

For small inputs, cost for adding objects just similar to the blocked set. However, cost for searching is much faster as retrieving data only need to retrieving the pointer address for the node.

Medium

```
Time cost for current set is 496
Invalid check: 2000: 0
Contain check: 1000: 1
Node removed for 1000.
Contain check: 1000: 0
The process costs 90 for removing and containing test
Process finished with exit code 0
```

For medium inputs, the conclusion is the same as that in the small number of inputs testing.

Large

```
Time cost for current set is 1869
Invalid check: 2000: 0
Contain check: 1000: 1
Node removed for 1000.
Contain check: 1000: 0
The process costs 112 for removing and containing test
Process finished with exit code 0
```

Adding large amount of data costs extremely lower than the blocked set. This is because that address are stored in the mapping header, however objects are created concurrently. Moreover, writing value to the pointer array is much more faster than writing a lot of stuff in the whole node. Hence the mapped set represents a higher efficiency than the blocked set.

Section 5: Conclusion

In this practical, there are two concurrent sets are implemented, which does allows being accessed by multiple threads. These concurrent sets uses locks for prevents the data race and protect the data. Since concurrency happens for high performance, preventing data race is critical for us to considers.

This practical have learnt me a lot. The text book introduces a non-blocking way of implementing concurrent sets, however due to the fever and cough, I am unable to implement that set for this practical. I would improve my self and try my best for our next session.