OS Assignment 4

Era Sarda, 2020MT10801
Ritika Soni, 2020MT10838

October 2023

PART 1:
Please Note : The bytes for "\n" new line character are getting added into the "Read bytes" or "Written bytes" on their own. Apart from this, our code is running fine.

PART 2

1. We assumed that no duplicate keys will be asked to be inserted in the AVL Tree.
2. We assumed all exit command will be given at the end of input file.
3. The behavior of an AVL tree is different from that in a single-threaded environment due to the race conditions and conflicts between multiple threads.
4. Concurrency introduces overhead compared to the traditional implementation. This overhead impacted performance.
5. The lock used in the AVL Tree is similar to the "read-write-lock" from the first part of the Assignment. The write part of the lock is used for insertion, deletion and inorder traversal (as we're allowing only one inorder traversal to be performed at a time). The read part of the lock is used for searching the key in the AVL Tree.
6. The search operation (contains) is read-only, and therefore, it is performed concurrently without synchronization, as it doesn't modify the tree structure. This lead to better parallelism. But it's paused when there's a write call (i.e. insertion or deletion call) waiting.
7. While concurrent AVL trees take advantage of parallelism for search operations (contains), insertions, deletions, and in-order traversals were more challenging and lead to contention.
8. For the same input, pre-order traversal printed different outputs. This was due the formation of different structure of AVL trees, which was in fact different due the nature of concurrent environment, the Threads and Lock acquisitions. For example see Fig. 4 as the input file and Fig. 5, Fig. 6 and Fig. 7 as their outputs sequences. In traditional AVL Tree only one preorder will be printed everytime.
9. For the same input, order of insertion of elements or deletion of elements may not be the same as provided by the input file (implementation in the same order as provided, happens in case of traditional AVL Trees). For example see Fig. 1 as the input file and Fig. 2 and Fig. 3 as their outputs sequences.
10. If more than one threads were allowed to run parallel for inorder, like the case in search, when we called inorder function, due to the recursion used in its implementation, the data being stored in the "inorder-array" was not in proper inorder pattern, as these threads were interupting each other.
11. If more than one threads were allowed to run parallel for inorder, they interrupted each other and printed wrong outputs while printing the output in the for loop.
12. If the thread of inorder completes before completion of the insert and delete functions present before it, in the input file, then inorder can print junk values.

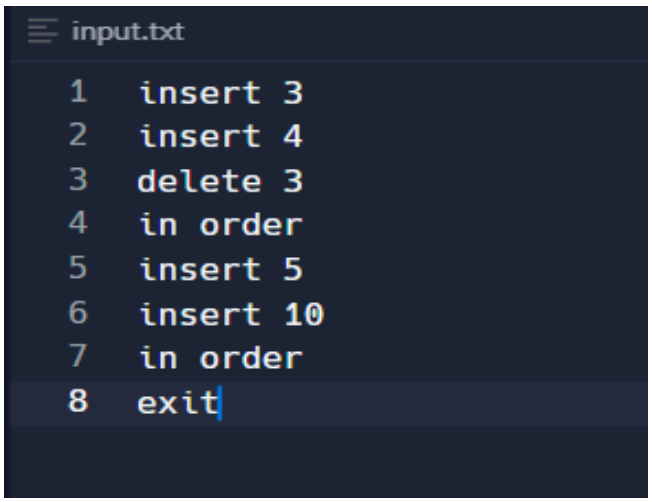

Figure 1: Input1

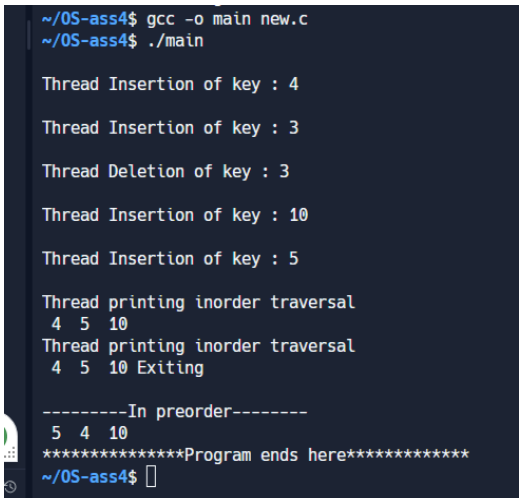

Figure 2: output1.1

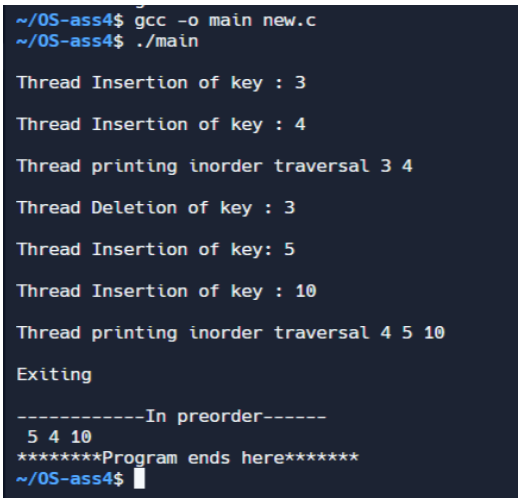

Figure 3: output1.2

input.txt

```
1   insert 5
2   insert 2
3   insert 7
4   insert 1
5   insert 3
6   delete 2
7   contains 7
8   insert 9
9   insert 6
10  in order
11  exit
12
```

Figure 4: Input2

```
~/OS-ass4$ gcc -o main 1.c
~/OS-ass4$ ./main
yes
1 3 5 6 7 9
6 3 1 5 7 9
~/OS-ass4$
```

Figure 5: output2.1

```
~/OS-ass4$ ./main
yes
1 3 5 6 7 9
5 1 3 7 6 9
~/OS-ass4$
```

Figure 6: output2.2

```
~/OS-ass4$ gcc -o main 1.c
~/OS-ass4$ ./main
1 3 5 6 7 9
yes
6 3 1 5 7 9
~/OS-ass4$
```

Figure 7: Output2.3