CS636 - Analysis of Concurrent Programs
2020-2021-II: Assignment 2

Name: Sarthak Singhal Roll No.: 170635

1. Problem 1

- How to compile:

```
g++ airlinebooking.cpp -pthread -latomic
```

- The header file *nb_queue.h* contains the class which defines the non blocking queue and its APIs enqueue, dequeue, dump_queue. The queue data structure is implemented using the given research paper. The object of queue in constructed inside *airlinebooking.cpp.* Also, all the thread creation work and the functions of customers and producer are present inside *airlinebooking.cpp.* The file *nb_queue.h* just provides the APIs for enqueue, dequeue, dump_queue.
- The producer thread enqueues the seats in increasing order.
- The customer threads generate a random number corresponding to number of seats to be booked and retries until either it books all its seats or there are no more available seats.
- The number of available seats is monitored using an atomic variable initialized to SEATS.
- After booking the seats the customer threads take a common print lock and outputs the numbers of the seats booked.
- The usage of dump_queue method is demonstrated inside *airlinebooking.cpp* inside comments after calls to enqueue and dequeue.
- The working of dump_queue is such that it waits if there are any pending enqueues and dequeues. The new enqueues and dequeues wait until the dump_queue method is complete.

2. **Problem 2**

- There are 3 files: *problem2.cpp*, *blocking_hashtable.h*, *nonblocking_hashtable.h*. The file *problem2.cpp* does all the work of thread creation, reading file and calling corresponding functions insert, remove and find. The remaining 2 files define the classes of blocking and non-blocking hash tables and also provide APIs for insert, remove and find.
- For part I, include the header *#include "blocking_hashtable.h"* inside *problem2.cpp*. While for part II, include the header *#include "nonblocking_hashtable.h"* inside *problem2.cpp*.
- How to compile:

```
g++ problem2.cpp -pthread -latomic
```

(a) Part I

- The data corresponding to each index is stored in the form of linked list.
- A vector of size equal to size of hash table is made which stores heads and tails of the corresponding linked lists.
- There are locks for each index to provide correctness. Also, there is 1 lock for printing.
- Each of the function insert, remove and find, first computes hash value of given word and takes lock corresponding to that index. Due to this only 1 thread works at a time on a particular hash bucket.
- find method takes the print lock when it outputs the sentences corresponding to the given word.

(b) Part II

- **NOTE:** I have discussed the approach towards the algorithm with Aniket Sanghi (170110) and Paramveer Raol(170459). We have not discussed the code or any implementation details. Also, the idea of the algorithm is taken from the nonblocking algorithm of Set data structure from the lectures. And, the idea of handling the ABA problem is taken from the research paper provided in problem 1.
- The data structures for linked list nodes are somewhat similar to the ones that were used for problem1. The *Pointer* class stores the address of next linked list node, a counter to handle ABA problem, and a boolean variable that stores whether the node is marked or not i.e. logically deleted or not. All the CAS operations are done on the objects of this class.
- find method:
  - Starts iterating from the head of corresponding linked list obtained using hash of the word.
  - If the first occurence of the word is marked, then it means either a remove had been called earlier or a remove is going in progress. So, find returns ENOWORD immediately.
  - Else it stores all the sentences till it reaches first marked node.
  - It then validates if the first occurence of the word in the linked list is currently marked or not. If it is marked then a remove has been started or completed. So, it

    returns ENOWORD.
  - After taking print lock it outputs all the sentences stored while iterating.

- remove method:
  - Starts iterating from the head of corresponding linked list obtained using hash of the word.
  - If the node of the word is marked, then it means either a remove had been called earlier or a remove is going in progress. So, remove returns ENOWORD immediately.
  - Else it sets the mark bit atomically for the node which has the given word.
  - Then moves to the next node and repeats the process.
- insert method:
  - Inserts the newnode in the start of the corresponding linked list atomically.
  - After inserting removes all the marked nodes from the linked list physically.