Dry 3

Ben Weiss – 326876752 – benweiss@campus.technion.ac.il
Itzik Solomon – 201522315 – topitzik@gmail.com

Question 1

1.
   a. Search:
      i. While i:
         1. Lock node i-1 (if exists)
         2. Lock node i
         3. … Do whatever
         4. Lock node i+1 (if exists)
         5. Unlock node i-1 (if exists)
         6. i <- i+1
   b. Insert (node j before node i, and assuming in search and node i-1 and node i are locked):
      i. Lock node j
      ii. Align pointers (i-1 -> j -> i)
      iii. Unlock node i-1
   c. Remove (node i, and assuming in search and node i-1 and node i are locked):
      i. Lock node i+1
      ii. Align pointers (i-1 -> i+1)
      iii. Free/return node i

2. Observation 1: Basically, the observation means that one thread (we'll denote as B) cannot overtake another thread (we'll denote as A). This is true because when a thread reads a node, it locks it, meaning no other thread can read the same node. Since a thread (A) always hold the lock for at least one node at any given time, in order for another thread (B) to overtake it (A), it (B) needs to go through a node that is currently locked by the first thread (A). This is obviously not possible since a locked node cannot be read and thus cannot be passed.

   Observation 3: When a thread is using a prime candidate *p*, all prime numbers less then *p* have already been used by the thread. Together with observation 1 we can infer that all non-prime numbers$\leq p$, which are a multiplication of the used primes, were deleted (by either this thread or another). p is not divisible by any previous number (or else it would have been deleted) $\Rightarrow p\ is\ prime$.

   Observation 2: Observation 3 $\Rightarrow p$ is prime $\Rightarrow p^2$ is only divisible by $p, 1 \Rightarrow p^2$ can only be removed by p.

3. The algorithm needs to delete all non-prime numbers. If all threads run the exact same (single-threaded with hand-over-hand synchronization) code, the lock becomes the

bottleneck. Only one thread, we'll denote as T1, will succeed in locking the first node (we'll denote as N) before all else, while all other threads will have to wait until that node is unlocked. Hand-over-hand dictates that T1 will always hold the lock of at least one node at any given time, so other threads will never be able to overtake it. T1 will always be first to arrive at each non-prime number, thus T will be the only thread to delete any number.

Question 2

1. For this algorithm, we'll use the upgrade_to_write_lock() function from Question 3 and another function, downgrade_to_read_lock(), which will be atomic like upgrade. Each node will have two counters and three locks – a counter for num_of_readers, num_of_writers, a lock for the counters, a lock for threads trying to upgrade, and a lock for threads trying to downgrade.
   Each thread will store the prime candidate its operated on (to help it know from where to start considering prime candidates), initialized to 0, as "last_candidate".
   Each thread will start at the head of the list, lock the first node (i=2) for reading, and will use it as a prime candidate. To iterate, it'll lock the next node for reading (i+1) and release the previous node (i-1), assuming each exist, hand-over-hand and set i to i+1. To delete node i, already holding read locks for i-1 and i, it'll release i and then try to upgrade i-1 to a write lock. If it fails (ie, another thread is also trying to upgrade to write), then abort, reset i to the head, and continue deleting back from there. After deleting the node, it releases i+1 and downgrades i-1 to a read lock. Once a thread finishes deleting all multiples of its prime candidate, it'll update last_candidate, reset i to the head, and then pick the next prime candidate as the next node that is larger than last_candidate. When deleting multiples, if it passes the prime candidate's square without having deleted it itself, the thread will stop and select another prime candidate as described. Note that we don't guarantee that a prime candidate is actually prime, but we try to delete its multiples anyways. Also note that 2 threads that are searching for multiples may pass each other due to the fact that they are both in read mode and neither blocks the other.
2. Example – Two threads (we'll call T1 and T2) and N=8. T1 starts at 2 and continues to delete 4. At this time, he holds a read lock for 3 (just downgraded from a write lock). Immediately after, T2 starts at 2, continues and realizes that 4 (=2^2) has been deleted, and looks for its next prime candidate. It selects 3 (the one directly after 2), continues and removes 6, reaches the end of the list (as 9 is out of the range), and handles the next prime candidate (5) and the next one (7). At this point, when it searches for the next prime candidate, it'll use 8, which isn't prime. Thus it is possible that a prime candidate that isn't prime might be chosen.

Question 3

1. Only one thread should be permitted to upgrade its reader state to a writer, because if two threads try to upgrade, they both will wait for the other thread to release the read lock, causing a deadlock. By making one fail, we indirectly releasing its lock, letting the other thread to successfully acquire the write lock.

2.
```
pthread_mutex_t upgrade_lock;

bool upgrade_to_write_lock(){
    int res = pthread_mutex_trylock(&upgrade_lock);
    if (res == EBUSY){
            return false;
    }
    pthread_mutex_lock(&global_lock);
    number_of_readers--;
    while ((number_of_writers > 0) || (number_of_readers > 0))
            pthread_cond_wait(&writers_condition, &global_lock);

    number_of_writers++;
    pthread_mutex_unlock(&upgrade_lock);
    pthread_mutex_unlock(&global_lock);
    return true;
}
```