Dry 3

Ben Weiss – 326876752 – [benweiss@campus.technion.ac.il](mailto:benweiss@campus.technion.ac.il)  
Itzik Solomon – 201522315 – [topitzik@gmail.com](mailto:topitzik@gmail.com)

# Question 1

* 1. Search:
     1. 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
  2. Insert (node j before node i, and assuming in search and node i-1 and node i are locked):
     1. Lock node j
     2. Align pointers (i-1 -> j -> i)
     3. Unlock node i-1
  3. Remove (node i, and assuming in search and node i-1 and node i are locked):
     1. Lock node i+1
     2. Align pointers (i-1 -> i+1)
     3. Free/return node i

1. 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, which are a multiplication of the used primes, were deleted (by either this thread or another). is not divisible by any previous number (or else it would have been deleted)   
   Observation 2: Observation 3 is prime is only divisible by can only be removed by
2. 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

באלגוריתם זה נשתמש ב upgrade\_to\_write\_lockמשאלה 3 ו-downgrade\_to\_write\_lock (יפעל בדומה לupgrade בכך שהפעולה היא אטומית. לכל צומת יש כעת שלושה מנעולים, הרגיל ושניים שטובת שתי הפונקציות הנ"ל.

כל חוט ישמור את המספר הגדול ביותר שהשתמש בו כמועמד לראשוני (last\_checked).

האלגוריתם יתחיל בראש הרשימה, ינעל את המספר הראשון(2) לקריאה וישתמש בו כמועמד להיות ראשוני. כאשר הוא רוצה להתקדם כדי לקרוא את הצומת ה-i+1 ברשימה, הוא ינעל לקריאה את הצומת ולאחר מכן ישחרר את הצומת i-1(hand-over-hand). אם החוט מצא שצריך למחוק את i+1 הוא ישחרר את האיבר הi+1 ויקרא ל upgrade\_to\_write\_lock(משאלה 3) על האיבר הi. אם upgrade\_to\_write\_lock נכשל, זה סימן שעוד חוט מנסה למחוק את אותו צומת ולכן במקרה נדיר זה נחזור לראש הרשימה. אם הצליח בשידרוג ימשיך לנעול לכתיבה(לא upgrade) את i+1 ו- i+2 וימחק את i+1. לאחר מכן יבצע downgrade\_to\_write\_lock לi וימשיך משם. אם חוט מגלה שהוא עבר את ריבוע המועמד, או שהגיע לסוף הרשימה, הוא יחזור לראש הרשימה. כדי שלא נעבור שוב על אותם צמתים, נדלג לצומת הראשון שגדול מ last\_checked.

\*\*\*\*\*\*\*\*\*\*\*

ישנם שני חוטים T1, T2. אנחנו רצים עד 8.

T1 מתחיל עם 2 ומוחק את 4. כעת T1 נעול לקריאה רק על 3 ונח על זרי הדפנה.

T2 מתחיל עם 2 ומגלה ש4 מחוק. לאחר מכן הוא מתחיל עם 3 ומגלה ש9 לא קיים. באותו אופן יעבור על 5 ו-7. לבסוף יעבור על 8 שהוא כמובן לא ראשוני.

# 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);

while ((number\_of\_writers > 0) || (number\_of\_readers > 1))

pthread\_cond\_wait(&writers\_condition, &global\_lock);

number\_of\_writers++;

pthread\_mutex\_unlock(&upgrade\_lock);

pthread\_mutex\_unlock(&global\_lock);

return true;  
}