Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχανικών Η/Υ και Πληροφορικής

ΜΥΥ601
Λειτουργικά Συστήματα
-2023-

Εργαστήριο 1:Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

ΘΩΜΑ ΕΙΡΗΝΗ 4292 , ΤΣΙΝΟΡΕΜΑΣ ΗΛΙΑΣ 2151, ΜΑΗ ΧΡΙΣΤΙΝΑ 4612

Σύντομη περιγραφή πρώτου βήματος ταυτοχρονισμού με χρήση μιας καθολικής κλειδαριάς:

Αρχικά, μας ζητήθηκε μια τετριμμένη υλοποίηση με την χρήση μιας κλειδαριάς, για να εφαρμοστεί αμοιβαίος αποκλεισμός στις λειτουργίες «put» και «get». Στόχος μας ήταν να μπορούν πολλά νήματα να κάνουν «put» και «get» χωρίς να υπάρξουν συγκρούσεις στις δομές και segmentation fault. Έτσι, αφού κατανοήσαμε σε γενικά πλαίσια τις λειτουργίες της μηχανής αποθήκευσης Kiwi αρχίσαμε τον πειραματισμό.

Προκειμένου να πετύχουμε ένα απλό επίπεδο πολυνηματισμού στην βάση αποφασίσαμε την δημιουργία καθολικής κλειδαριάς στις μεθόδους «db_add()» και «db_get()» ώστε να γίνεται μία λειτουργία την φορά. Αυτή η σκέψη λειτουργεί ως εξής: κάθε φορά που θα γίνεται ένα «put» ή «get» το lock να κλειδώνεται για να πετύχουμε έναν αμοιβαίο αποκλεισμό μεταξύ των μεθόδων , διασφαλίζοντας ότι οι δύο μέθοδοι δεν θα συμβούν ταυτόχρονα και αποκλείοντας την περίπτωση σφάλματος. Βέβαια η υλοποίηση αυτή δεν μας προσφέρει ένα καλό επίπεδο ταυτοχρονισμού καθώς εκτελείται μόνο ένα «get» ή ένα «put» την φορά , ασχέτως του πλήθους των νημάτων.

Για την υλοποίηση χρειάστηκε να παρέμβουμε σε πέντε αρχεία, το bench.c, το bench.h, το db.c, το db.h και το kiwi.c.

Τροποποιήσεις και πρόσθετα:

Bench.c

```
int main(int argc,char** argv)
{
    long int count;
    int num_threads; /*input number of threads from the keyboard*/
    pthread_mutex_init(&glob_lock, NULL); /*lock initialization  --> extern in
db.h*/
    srand(time(NULL));
```

Αρχικά , ξεκινήσαμε αρχικοποιώντας μια κλειδαριά - mutex - η οποία κάνει extern στο αρχείο db.h (→db.h : `extern pthread_mutex_t glob_lock;  /*declare to mutex*/`----έχει προστεθεί στο db.h), και δίνοντας το πλήθος των νημάτων από το πληκτρολόγιο.

```
/*
changing argc < 4 because the number of inputs increased.
input0: ./kiwi-bench, input1: write | read, input2: count, input3: num threads
*/
    if (argc < 4) {  /**/
        fprintf(stderr,"Usage: db-bench <write | read> <count> <num
threads>\n");   /**/
        exit(1);
    }

        if (strcmp(argv[1], "write") == 0) {
        int r = 0;
        count = atoi(argv[2]);
```

```
        num_threads = atoi(argv[3]); /*how many threads it will get from the
keyboard*/
        _print_header(count);
        _print_environment();
        if (argc == 5) /**/
            r = 1;
        thread_write(count, r, num_threads); /*extra argument the num_threads*/
    } else if (strcmp(argv[1], "read") == 0) {
        int r = 0;
        count = atoi(argv[2]);
        num_threads = atoi(argv[3]); /*threads from the keyboard*/
        _print_header(count);
        _print_environment();
```

Στο παραπάνω κώδικα βλέπουμε την περίπτωση στην οποία ο χρήστης θελήσει να γράψει στην βάση δεδομένων. Παρατηρούμε ότι το `if (argc < 4)` άλλαξε σε 4 από 3 καθώς στο fprintf προστέθηκε ένα 4ο όρισμα , το `<num threads>` δηλαδή η επιλογή του πλήθους των νημάτων που θα δημιουργηθούν για να εκτελέσουν τις εγγραφές στην βάση. Έπειτα, υπάρχει το `num_threads = atoi(argv[3])` όπου παίρνουμε το πλήθος των νημάτων κάνοντας κλήση της atoi() η οποία μετατρέπει ένα string σε ακέραιο. Η _write_test που προϋπήρχε έγινε thread_write με ένα επιπλέον όρισμα, το num_threads.

```
/*
changing argc == 5 because the number of inputs increased.
input0: ./kiwi-bench, input1: write|read, input2: count, input3: num threads,
input4: random
*/
        if (argc == 5)  /*4 -> 5*/
            r = 1;
        thread_read(count, r, num_threads);  /*extra argument the num_threads*/
    } else {
        fprintf(stderr,"Usage: db-bench <write | read> <count> <num threads>
<random>\n");/**/
        exit(1); /**/
    }
    return 1;
}
```

Παρομοίως με το προηγούμενο κομμάτι κώδικα , εδώ βλέπουμε την περίπτωση στην οποία ο χρήστης θελήσει να διαβάσει στην βάση δεδομένων. Κι εδώ άλλαξε το «`argc == 5`» από 4 σε 5 καθώς προστέθηκε ένα επιπλέον όρισμα στο fprintf (το `<num threads`) δηλαδή η επιλογή του πλήθους των νημάτων που θα δημιουργηθούν για να εκτελέσουν τις αναγνώσεις στην βάση. Η `_read_test` που προϋπήρχε έγινε `thread_read` με ένα επιπλέον όρισμα, το `num_threads`.

Bench.h

```
/*
we are making threads in kiwi.c, and extern in db.h so we use the same
headerfiles
*/
#include "../engine/db.h" /**/
#include "../engine/variant.h"  /**/
…
…
/*
a struct we created to pass the needed data to the _write_test,
thread_write, _read_test functions and thread_read  --> kiwi.c
*/
typedef struct paramet{
    long int count; /*copy form kiwi.c*/
    int r; /*copy form kiwi.c*/
    DB *db; /*copy form kiwi.c*/
}paramet;


void thread_write(long int count, int r, int num_threads); /*declare
thread_write fucntion --> kiwi.c*/
void thread_read(long int count, int r, int num_threads); /*declare
thread_read fucntion --> kiwi.c*/
```

Σε αυτό το σημείο , βλέπουμε την προσθήκη των include τα οποία υπάρχουν ήδη και σε άλλα αρχεία όπως το kiwi.c καθώς παρατηρήσαμε ότι περιέχουν κάποιες βιβλιοθήκες που θα χρειαστούμε, πχ την threads.h . Επίσης χρησιμοποιείται και για το extern στο db.h.
Πιο απλά, δεν χρειαζόμαστε το include <pthreads.h> γιατί όταν κάνεις τα νήματα στο kiwi.c παίρνει header file το db.h όπου έχουν οριστεί ήδη εκεί τα threads (με include <pthreads.h>). 'Ετσι προσθέσαμε τα ίδια header file και στο bench.h καθώς στο bench.c όπως είδαμε και προηγουμένως έχουμε αρχικοποιήσει το mutex.

Στην συνέχεια , έχουμε υλοποιήσει ένα struct από το οποίο θα στο οποίο έχουμε αντιγράψει και προσθέσει μεταβλητές από το kiwi.c.

Έχουμε δηλώσει τις συναρτήσεις thread_write, thread_read.

Db.c

```
/*
we put lock-unlock locks ---ret_value is a helper
variable because of the return statement-----
*/
```

```
int db_add(DB* self, Variant* key, Variant* value)
{
    int ret_value; /**/
    pthread_mutex_lock(&glob_lock); /**/
    if (memtable_needs_compaction(self->memtable))
    {
        INFO("Starting compaction of the memtable after %d insertions and %d
deletions",
             self->memtable->add_count, self->memtable->del_count);
        sst_merge(self->sst, self->memtable);
        memtable_reset(self->memtable);
    }
    ret_value = memtable_add(self->memtable, key, value); /**/
    pthread_mutex_unlock(&glob_lock); /**/
    return ret_value; /**/
}
/*
we put lock-unlock locks ---ret_value is a helper
variable because of the return statement-----
*/
int db_get(DB* self, Variant* key, Variant* value)
{
    int ret_value; /**/
    pthread_mutex_lock(&glob_lock); /**/
    if (memtable_get(self->memtable->list, key, value) == 1)
        return 1;
    ret_value = sst_get(self->sst, key, value); /**/
    pthread_mutex_unlock(&glob_lock); /**/
    return ret_value; /**/
}
```

Στο συγκεκριμένο αρχείο υλοποιήσαμε κλειδαριά lock-unlock στο db_add (-> writers) και στο db_get (->readers). Κάνουμε lock στην αρχή κάθε μεθόδου και unlock στο τέλος. Επειδή η κλειδαριά lock-unlock κάνει return απευθείας την συνάρτηση δεν γίνεται να κάνουμε το mutex κάτω από το return statement καθώς θα «φύγει» από την συνάρτηση, για αυτό το αναθέτουμε σε μια βοηθητική μεταβλητή ret_value ώστε να αποθηκεύεται η τιμή στην μεταβλητή, να κάνει unlock , και στην συνέχεια να επιστρέφει και να κλείνει η συνάρτηση (lock).

Kiwi.c

```
int found = 0; /*global*/

/*-------------------------------write-----------------------------------*/

void* _write_test(void *x) /* change of _write_test function*/
{
```

```
…
…
    paramet param = *(paramet *)x; /* cast pointer x to (paramet *) */
…
…
    for (i = 0; i < param.count; i++) { /*same* + count changed to
param.count*/
        if (param.r)  /* r changed to param.r  (struct in bench.h)*/
…
…
        db_add(param.db, &sk, &sv); /*same + db changed to param.db (struct in
bench.c)*/
        if ((i % 10000) == 0) { /*same*/
            fprintf(stderr,"random write finished %d ops%30s\r",
                    i,
                    ""); /*same*/
            fflush(stderr); /*same*/
        }
    }
    return 0; /**/
}
```

Σε αυτό το κομμάτι δεν έγινε κάποια συνταρακτική αλλαγή , το μεγαλύτερο κομμάτι του κώδικα παραμένει ίδιο με διαφορά την σύνδεση του struct (->param) με τις μεταβλητές count, r, db. Επίσης κάναμε global το found . . Επίσης επειδή βάλαμε τα threads μέσα σε συνάρτηση στην thread_write, έγινε `void* _write_test(void *x)`.

```
void thread_write(long int count, int r, int num_threads){ /*prototype for
thread_write function*/
    double cost; /*same*/
    long long start,end; /*same*/
    paramet par; /*declare struct*/
    pthread_t th[num_threads]; /*initialize threads*/
    par.count = count; /**/
    par.r = r; /**/
    par.db = db_open(DATAS); /*db changed to par.db (struct in bench.h)*/
    start = get_ustime_sec(); /*same*/
/*
creating threads in a loop
*/
    for (int i=0; i<num_threads; i++){
        if (pthread_create(&th[i], NULL, &_write_test, &par) != 0){ /*creating
the thread*/
            fprintf(stderr,"error creating thread\n");
            exit(1);
        }
```

```
    }
/*
joinning created threads in a loop
*/
    for (int i=0; i<num_threads; i++){
        if (pthread_join(th[i], NULL) != 0){ /*joinning the thread*/
            fprintf(stderr,"error join thread\n");
            exit(1);
        }
    }
    db_close(par.db); /*changed db to par db (struct bench.h)*/
    end = get_ustime_sec(); /*same*/
    cost = end -start; /*same*/
    printf(LINE); /*same*/
    printf("|Random-Write   (done:%ld): %.6f sec/op; %.1f
writes/sec(estimated); cost:%.3f(sec);\n"
        ,par.count, (double)(cost / par.count)
        ,(double)(par.count / cost)
        ,cost); /*same + changed count to par.count (struct in bench.h)*/
}
```

Παραπάνω βλέπουμε την διαδικασία υλοποίησης των νημάτων για την write.

```
/*--------------------------------read----------------------------------*/

void* _read_test(void *x) /*change of _read_test function*/
{
…
…
    paramet param = *(paramet *)x; /* cast pointer x to (paramet *) */

    char key[KSIZE + 1];

    for (i = 0; i < param.count; i++) { /*same + changed count to param.count
(struct in bench.h)*/
        memset(key, 0, KSIZE + 1);/*same*/

        /* if you want to test random write, use the following */
        if (param.r) /*changed r to param.r (struct in bench.h)*/
            _random_key(key, KSIZE); /*same*/
        else
            snprintf(key, KSIZE, "key-%d", i); /*same*/

        fprintf(stderr, "%d searching %s\n", i, key);
        sk.length = KSIZE; /*same*/
        sk.mem = key; /*same*/
```

```
        ret = db_get(param.db, &sk, &sv); /*changed db to param.db (struct in
bench.h)*/
…
…
```

Σε αυτό το κομμάτι δεν έγινε κάποια συνταρακτική αλλαγή , το μεγαλύτερο κομμάτι του κώδικα παραμένει ίδιο με διαφορά την σύνδεση του struct (->param) με τις μεταβλητές count, r, db. Επίσης επειδή βάλαμε τα threads μέσα σε συνάρτηση στην thread_read, έγινε `void* _read_test(void *x).`

```
void thread_read(long int count, int r, int num_threads){ /*prototype for
thread_read function*/

    long long start,end; /*same*/
    double cost; /*same*/
    int remain, quotient; /*declare variables*/
    DB *db; /*same*/
    paramet par, par2; /*declare struct*/
    pthread_t th[num_threads];  /*initialize threads*/
    remain = count % num_threads; /*the remainder of count / threads*/
    quotient = count / num_threads; /*the quotient of count / threads*/
    par.count = quotient; /**/
    par.r = r; /**/
    par2.r = r; /**/
    par2.count = quotient + 1; /**/
…
…
/*
creating threads in a loop
*/
    for (int i=0; i<num_threads; i++){
/*
if remain > 0 . distribute the remainder into as many
threads as needed so that remain = 0
example : 10 reads to 6 threads --> remain = 4 , so thread1=1 +1, thread2=1 +1,
thread3=1 +1, thread4=1 +1, thread5=1, thread6=1,*/
        if (remain > 0){ /**/
            remain--; /**/
            if (pthread_create(&th[i], NULL, &_read_test, &par2) != 0){
/*creating the thread*/
                fprintf(stderr,"error creating thread\n");
                exit(1);
            }
        }
        else{
            if (pthread_create(&th[i], NULL, &_read_test, &par) != 0){
/*creating the thread*/
```

```
                fprintf(stderr,"error creating thread\n");
                exit(1);
            }
        }
    }
/*
joinning created threads in a loop
*/
    for (int i=0; i<num_threads; i++){
        if (pthread_join(th[i], NULL) != 0){
            fprintf(stderr,"error join thread\n"); /*joinning the thread*/
            exit(1);
        }
    }
……
    printf("|Random-Write   (done:%ld): %.6f sec/op; %.1f
writes/sec(estimated); cost:%.3f(sec);\n"
        ,par.count, (double)(cost / par.count)
        ,(double)(par.count / cost)
        ,cost); /*same + changed count to par.count (struct in bench.h)*/
```

Παραπάνω βλέπουμε την διαδικασία υλοποίησης των νημάτων για την read.
Επιπλέον, στην thread_read γίνεται ο διαμοιρασμός στα νήματα. Η λογική της υλοποίησης είναι η εξής: Αρχικά υπήρχε ο προβληματισμός πως θα μοιραστούν σωστά και δίκαια τα δεδομένα στα νήματα. Το πρόβλημα υπήρχε στις περιπτώσεις που τα δεδομένα έκαναν διαίρεση με τα νήματα αλλά είχαμε ως αποτέλεσμα δεκαδικό (πχ 5 δεδομένα σε 2 νήματα μας δίνει 2,5 σε κάθε νήμα). Η σκέψη μας ήταν να γίνεται ακέραια διαίρεση, το πηλίκο να διαμοιράζεται το ίδιο στα νήματα και το υπόλοιπο να προστίθεται σε ένα από τα νήματα στο τέλος, πχ 5 δεδομένα σε δύο νήματα, σε ακέραια διαίρεση μας δίνει υπόλοιπο 1 και πηλίκο 2, άρα το 1ο νήμα θα πάρει 2, το δεύτερο άλλα 2 και το υπόλοιπο 1 θα προστεθεί ή στο 1ο ή στο 2ο, άρα 3 στο ένα νήμα και 2 στο άλλο. Μετά συνειδητοποιήσαμε ότι το υπόλοιπο δεν θα είναι πάντα 1 , πχ το read 100 80 μας δίνει 100 δεδομένα σε 80 νήματα που σημαίνει ότι έχουμε υπόλοιπο 20 δεδομένα. Αρχικά για να λύσουμε το πρόβλημα σκεφτήκαμε την χρήση ενός πίνακα , αλλά καταλήξαμε στην παραπάνω υλοποίηση.

Μια πρώτη προσπάθεια για την υλοποίηση ταυτοχρονισμού με καθολική κλειδαριά ήταν η δημιουργία δύο ενδιάμεσων συναρτήσεων στο αρχείο bench.c οι οποίες δεν έκανα διαμοιρασμό των νημάτων όπως θα θέλαμε αλλά γέμιζε το κάθε νήμα με εγγραφές όσες και οι εγγραφές που έβαζε ο χρήστης. Για παράδειγμα, εάν ο χρήστης έδινε από το πληκτρολόγιο «writes 10 2» , αντί να δημιουργούνται 2 νήματα με 5 εγγραφές το κάθε ένα, γίνονταν 2 νήματα με 10 εγγραφές το κάθε ένα

Σύντομη περιγραφή <u>δεύτερου βήματος</u> ταυτοχρονισμού με χρήση αλγορίθμου γραφέων - αναγνωστών:

Για το δεύτερο βήμα ταυτοχρονισμού, μας ζητήθηκε μια βελτίωση της πρώτης υλοποίησης η οποία θα επιτρέπει πολλαπλούς αναγνώστες ή ένα γραφέα να λειτουργεί κάθε φορά.
Έτσι, αφού στο βήμα δημιουργήθηκαν πολλαπλά νήματα , τώρα θα χρειαστεί να δημιουργήσουμε την εντολή readwrite.

Σε αυτό το βήμα βασιστήκαμε στις αλλαγές που είχαμε ήδη κάνει για το πρώτο βήμα , προσθέτοντας μερικές ακόμη και αλλάζοντας άλλες.

Παρακάτω θα αναφερθούν και θα παρουσιαστούν μόνο τα κομμάτια που έχουν αλλαγές σε σχέση με το πρώτο βήμα.

Σε γενικά πλαίσια , (μικρό-)αλλαγές (συγκριτικά με το 1ο βήμα) μπορούν να παρατηρηθούν στα bench.c, db.c, db.h, kiwi.c. --Η κύρια αλλαγή έχει γίνει στα αρχεία db.c και bench.c

Αξίζει να αναφερθεί ότι για την υλοποίηση αυτή , βασιστήκαμε στο βιβλίο «Λειτουργικά Συστήματα -- Abraham Silberschatz».
Το οποίο μας βοήθησε να καταλάβουμε και να εφαρμόσουμε τον αλγόριθμο γραφέων-αναγνωστών με προτεραιότητα στους αναγνώστες.

Αναλυτικά:

Bench.c

```
int main(int argc,char** argv)
{
    long int count;
    int num_threads; /*input number of threads from the keyboard*/
    pthread_mutex_init(&glob_lock, NULL); /*dynamic lock initialization  -->
extern in db.h*/
    pthread_cond_init(&reader_wait, NULL); //*dynamic intialization --> extern
in db.h*/
    pthread_cond_init(&writer_wait, NULL); //*dynamic intialization --> extern
in db.h*/
    srand(time(NULL));
```

Στο συγκεκριμένο κομμάτι κώδικα, βλέπουμε ότι προστέθηκαν μερικές αρχικοποιήσεις για την προϋπόθεση ότι ο read ή ο write περιμένει.

```
……
/********* thread_write  →   ΊΔΙΟ ΜΕ ΤΗΝ ΠΡΏΤΗ ΥΛΟΠΟΊΗΣΗ **********/
……
        thread_read(count, r, num_threads);
/*
creating a readwrite option for read and write
requests from the same thread-s
*/
    } else if (strcmp(argv[1], "readwrite") == 0){
```

```
        int r = 0;
        double perce, count_r, count_w;
        count = atoi(argv[2]); /*total number of reads and writes*/
        num_threads = atoi(argv[3]);/* number of threads*/
        perce = atoi(argv[4]); /* from 1 to 100 */
        _print_header(count);
        _print_environment();
/*
argc == 6 because the number of inputs increased.
input0: ./kiwi-bench, input1: write|read|readwrite, input2: count, input3: num
threads, input4: random, input5: perce -only for readwrite-.
*/
        if (argc == 6){
            r = 1;
        }
        count_r = count * (perce / 100); /* the percentage of reads we want */
        count_w = count - (int)count_r;  /*remaining amount - rounding with the
"int",example: 32% for 7 = 2.24 -> round to 2 & the remain = int */
        thread_read((int)count_r, r, num_threads); /**/
        thread_write((int)count_w, r, num_threads); /**/
    } else {
        fprintf(stderr,"Usage: db-bench <write | read | readwrite> <count> <num
threads> <random>\n" "db-bench <readwrite> <count> <num threads> <percentage>
<random>\n");
        exit(1);
    }
```

Εδώ , συγκριτικά με το πρώτο βήμα, έχει αλλάξει ριζικά η thread_read καθώς πλέον υλοποιεί και την readwrite. Προσθέσαμε ενα else-if για την readwrite, το οποίο έχει μια μεταβλητή perce(percentage) που ο χρήστης επιλέγει το ποσοστό που θα πάρει η read, το ποσοστό που μένει πάει αυτόματα στη write (π.χ. αν ο χρήστης γρέψει 70 τότε η read θα πάρει το 70% των count και το υπόλοιπο 30% θα το πάρει η write). Το argc έπρεπε να γίνει 6 διότι προσθέσαμε ένα όρισμα παραπάνω στη γραμμώ εντολών (perce). Επίσης προσθέσαμε 2 double μεταβλητές το count_r και το count_w, επειδή η πράξη count*(perce/100) θα μας δώσει δεκαδικό αριθμό κάποιες φορές. Στην επόμενη γραμμή κάναμε cast σε int το count_r γιατί θέλαμε να πάρουμε το ακέραιο μερος του αριθμού (ουσιαστικά θέλαμε το floor του αριθμού γιατί δεν μπορούμε να έχουμε δεκαδικό). Στις συναρτήσεις thread_write, thread_read κάνουμε cast σε int γιατί παίρνουν ως ορίσματα ακεραίους. Τέλος έχουμε αλλάξει τις printf για να δείξουμε πιο καλά τη νέα λειτουργία readwrite.

```
    pthread_mutex_destroy(&glob_lock); /*destroy due to dynamic
initialization*/
    pthread_cond_destroy(&reader_wait); /*destroy due to dynamic
initialization*/
    pthread_cond_destroy(&writer_wait); /*destroy due to dynamic
initialization*/
    return 1;
}
```

Προστέθηκαν τα destroy.

Db.h

```
extern pthread_mutex_t glob_lock;  /*declare to mutex*/
extern pthread_cond_t reader_wait; /*declare*/
extern pthread_cond_t writer_wait; /*declare*/
```

Προστέθηκαν τα extern.

Db.c

```
#include <stdio.h> /**/
pthread_mutex_t glob_lock; /**/
/*
a condition to check
if readers are waiting or writers are waiting
*/
pthread_cond_t reader_wait, writer_wait;
/*
readers in waiting,  writers in waiting,
readers is active,  writers is active
*/
int wait_r, wait_w, act_r, act_w;
....
```

Στο db.c ξεκινάμε την υλοποίηση του αλγορίθμου γραφέων – αναγνωστών . Παραπάνω έχουμε ορίσει τις μεταβλητές για τους ενεργούς readers και writers, και τους readers και writers οι οποίοι περιμένουν. Βλέπουμε και το pthread_cond το οποίο μας βοηθά να ελέγχουμε την εκάστοτε κατάσταση.

```
....
/*
we put lock-unlock locks ---ret_value is a helper
variable because of the return statement-----
*/
int db_add(DB* self, Variant* key, Variant* value)
{
    int ret_value; /**/
    pthread_mutex_lock(&glob_lock); /**/
    while (act_w + act_r > 0){ /*check readers and writers*/
        wait_w++; /*increase number of writers waiting*/
        pthread_cond_wait(&writer_wait, &glob_lock);/**/
        wait_w--; /*decrease writers waiting*/
    }
    act_w++; /*  writer is active*/
    pthread_mutex_unlock(&glob_lock);/**/
    if (memtable_needs_compaction(self->memtable))
    {
```

```
        INFO("Starting compaction of the memtable after %d insertions and %d
deletions",
             self->memtable->add_count, self->memtable->del_count);
        sst_merge(self->sst, self->memtable);
        memtable_reset(self->memtable);
    }

    ret_value = memtable_add(self->memtable, key, value); /**/
    pthread_mutex_lock(&glob_lock); /**/
    act_w--; /*  writer not active*/
/*
if one or more readers are waiting, tell them to start
*/
    if (wait_r > 0){ /**/
        pthread_cond_broadcast(&reader_wait);
    }
 /*
 if someone writer is waiting, tell him to start
 */
    else if (wait_w > 0){
        pthread_cond_signal(&writer_wait);
    }
    pthread_mutex_unlock(&glob_lock);/**/
    return ret_value; //
}
```

Ελέγχουμε στην db_add εάν υπάρχει κάποιος ενεργός writer και reader, και αν τελικά υπάρχει κάνει wait και σταματάει στην pthread_cond_wait. Αφού περάσουμε απ' την κρίσιμη περιοχή ελέγχουμε αν υπάρχουν reader σε αναμονή, τότε στέλνουμε σήμα να ξεκινήσουν αλλιώς αν υπάρχει writer σε αναμονή στέλνει σε αυτόν σήμα να ξεκινήσει. Το wait ++ σημαίνει ότι περιμένει και το wait - - ότι δεν περιμένει. Αντίστοιχα και τα act ++ σημαίνει ότι είναι ενεργό και το act - - ότι δεν είναι.
Στο else-if δίνουμε προτεραιότητα στους readers που περιμένουν στέλνοντας μήνυμα -εάν περιμένουν- να ξεκινήσουν, ενώ εάν έχουμε writers που περιμένουν , στέλνουμε signal.
***κάνουμε broadcast αντί για signal γιατί οι readers μπορούν να μπουν όλοι μαζί ενώ οι writers ένας-ένας.

```
/*
we put lock-unlock locks ---ret_value is a helper
variable because of the return statement-----
*/
int db_get(DB* self, Variant* key, Variant* value)
{
    int ret_value; /**/
    pthread_mutex_lock(&glob_lock);  /**/
    while (act_w > 0){ /*check only the writers*/
        wait_r++; /* all readers in waiting*/
        pthread_cond_wait(&reader_wait, &glob_lock);/**/
        wait_r--; /* all readers in no waiting*/
```

```
    }
    act_r++; /*  readers is active, "++" -> another
    thread has entered */
    pthread_mutex_unlock(&glob_lock);
    if (memtable_get(self->memtable->list, key, value) == 1){
        ret_value = 1; /**/
    }
    else{
        ret_value = sst_get(self->sst, key, value); /**/
    }
    pthread_mutex_lock(&glob_lock); /**/
    act_r--; /* all readers is not active*/
    /*
    if we don't have active readers but we have at least one(or and more)
    active writer, send signal one to start
    */
    if (act_r == 0 && wait_w > 0){
        pthread_cond_signal(&writer_wait); /**/
    }
       pthread_mutex_unlock(&glob_lock); /**/
    return ret_value; /**/
}
```

Στην db_get ελέγχουμε μόνο τους writers, γιατί και να υπάρχει reader δεν μας επηρεάζει. Οπότε εξετάζει εάν είναι ενεργός ο writer, εάν είναι περιμένει – εάν δεν είναι προχωράει. Συνεχίζουμε μετά την κρίσιμη περιοχή, εάν δεν υπάρχει κανένας ενεργός read αλλά υπάρχει τουλάχιστον ένας writer που περιμένει ενεργοποίησε έναν από αυτούς.

Kiwi.c

```
void thread_write(long int count, int r, int num_threads){ /*prototype for
thread_write function*/
    long long start,end; /*same*/
    double cost; /*same*/
    int remain, quotient; /**/
    DB *db; /*same*/
    paramet par, par2; /*declare struct*/
    pthread_t th[num_threads]; /*initialize threads*/
    remain = count % num_threads; /*the remainder of count / threads*/
    quotient = count / num_threads; /*the quotient of count / threads*/
    par.count = quotient; /**/
    par2.count = quotient + 1; /**/
    par.r = r; /**/
    par2.r = r; /**/
…
/*
creating threads in a loop
```

```
*/
    for (int i=0; i<num_threads; i++){
/*
if remain > 0 . distribute the remainder into as many
threads as needed so that remain = 0
example : 10 reads to 6 threads --> remain = 4 , so thread1=1 +1, thread2=1 +1,
Thread3 = 1 +1, thread4 = 1 +1, thread5 = 1, thread6 = 1,
*/
        if (remain > 0){ /**/
            remain--; /**/
            if (pthread_create(&th[i], NULL, &_write_test, &par) != 0){
/*creating the thread*/
                fprintf(stderr,"error creating thread\n");
                exit(1);
            }
        }
        else{
            if (pthread_create(&th[i], NULL, &_write_test, &par) != 0){ /**/
                fprintf(stderr,"error creating thread\n"); /**/
                exit(1); /**/
            }
        }
    }
/*
join threads in a loop
*/
    for (int i=0; i<num_threads; i++){
        if (pthread_join(th[i], NULL) != 0){ /*joinning the thread*/
            fprintf(stderr,"error join thread\n");
            exit(1);
        }
    }

    db_close(par.db); /*changed db to par db (struct bench.h)*/
……
    printf("|Random-Write   (done:%ld): %.6f sec/op; %.1f
writes/sec(estimated); cost:%.3f(sec);\n"
        ,par.count, (double)(cost / par.count)
        ,(double)(par.count / cost)
        ,cost); /*same + changed count to par.count (struct in bench.h)*/
```

Για το kiwi.c η μόνη διαφορά είναι ότι η thread_write , έγινε ακριβώς όπως η thread_read στην 1η υλοποίηση.

Οδηγίες:

Βήμα 1: Αφού πλοηγηθούμε στον κατάλογο ~/kiwi/kiwi-source$ κάνουμε make clean εάν θέλουμε να σβήσουμε τα ήδη υπάρχον δεδομένα -εάν υπάρχουν-. Θα εμφανιστεί το παρακάτω μήνυμα:

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make clean
cd engine && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
rm -rf *.o libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
rm -f kiwi-bench
rm -rf testdb
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$
```

Βήμα 2: Για να τρέξουμε το πρόγραμμα θα χρειαστεί να κάνουμε make all στον κατάλογο ~/kiwi/kiwi-source$ και θα πρέπει να εμφανιστεί το παρακάτω:

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
    CC db.o
    CC memtable.o
    CC indexer.o
    CC sst.o
    CC sst_builder.o
    CC sst_loader.o
    CC sst_block_builder.o
    CC hash.o
    CC bloom_builder.o
    CC merger.o
    CC compaction.o
    CC skiplist.o
    CC buffer.o
    CC arena.o
    CC utils.o
    CC crc32.o
    CC file.o
    CC heap.o
    CC vector.o
    CC log.o
    CC lru.o
    AR libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
gcc -g -ggdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
```

Βήμα 3: Πηγαίνουμε στον κατάλογο ~/kiwi/kiwi-source/bench$ και είμαστε έτοιμοι να κάνουμε τα tests μας εφαρμόζοντας:

- ./kiwi-bench write [number of writes] [number of threads]
- ./kiwi-bench reads [number of reads] [number of threads]
- ./kiwi-bench readwrite [number of inputs] [number of threads] [percentage of reads]

Δοκιμές- παραδείγματα:

Write

Παραδειγμα 1

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 10 2
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        10
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Sat Apr  1 16:09:40 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[3054] 01 Apr 16:09:40.627 . file.c:200 Creating directory structure: testdb/si
[3054] 01 Apr 16:09:40.628 - file.c:211  -> Creating testdb
[3054] 01 Apr 16:09:40.635 - file.c:211  -> Creating testdb/si
[3054] 01 Apr 16:09:40.636 . sst.c:283 Manifest file not present
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
[3054] 01 Apr 16:09:40.643 . db.c:36 Closing database 10
```

Παραδειγμα 2

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 20 3
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        20
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Sun Apr  2 21:24:34 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1419] 02 Apr 21:24:34.127 . file.c:200 Creating directory structure: testdb/si
[1419] 02 Apr 21:24:34.128 - file.c:211  -> Creating testdb
[1419] 02 Apr 21:24:34.128 - file.c:211  -> Creating testdb/si
[1419] 02 Apr 21:24:34.128 . sst.c:283 Manifest file not present
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
[1419] 02 Apr 21:24:34.129 . db.c:36 Closing database 20
```

Παραδειγμα 3

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 15 7
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        15
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------
Date:           Sun Apr  2 21:25:58 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1437] 02 Apr 21:25:58.011 . file.c:200 Creating directory structure: testdb/si
.........
[1437] 02 Apr 21:25:58.013 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
[1437] 02 Apr 21:25:58.014 . db.c:36 Closing database 15
```

Read

Παραδειγμα 1

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 8 4
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        8
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------
Date:           Sat Apr  1 16:10:14 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[3060] 01 Apr 16:10:14.378 . file.c:200 Creating directory structure: testdb/si
```

.....

```
[3060] 01 Apr 16:10:14.379 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0 -- 139899415496448
1 searching key-1 -- 139899415496448
0 searching key-0 -- 139899423889152
1 searching key-1 -- 139899423889152
0 searching key-0 -- 139899432281856
1 searching key-1 -- 139899432281856
0 searching key-0 -- 139899440674560
1 searching key-1 -- 139899440674560
[3060] 01 Apr 16:10:14.379 . db.c:36 Closing database 0
```

Παραδειγμα 2

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 17 3
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        17
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------
Date:           Sun Apr  2 21:29:43 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1448] 02 Apr 21:29:43.430 . file.c:200 Creating directory structure: testdb/si
```

.....

```
[1448] 02 Apr 21:29:43.432 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
[1448] 02 Apr 21:29:43.433 . db.c:36 Closing database 0
```

Παραδειγμα 3

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 29 5
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        29
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------
Date:           Sun Apr  2 21:31:11 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1453] 02 Apr 21:31:11.003 . file.c:200 Creating directory structure: testdb/si
[1453] 02 Apr 21:31:11.005 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
5 searching key-5
[1453] 02 Apr 21:31:11.006 . db.c:36 Closing database 0
```

Read-write

Παραδειγμα 1

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 10 2 20
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        10
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Sat Apr  1 16:11:01 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

count r: 2, count w: 8
[3078] 01 Apr 16:11:01.585 . file.c:200 Creating directory structure: testdb/si
```

.....

```
[3078] 01 Apr 16:11:01.586 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0 -- 139836891518720
0 searching key-0 -- 139836899911424
[3078] 01 Apr 16:11:01.586 . db.c:36 Closing database 0
```

.....

```
[3078] 01 Apr 16:11:01.587 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
[3078] 01 Apr 16:11:01.587 . db.c:36 Closing database 8
[3078] 01 Apr 16:11:01.587 . sst.c:595 IN sst_merge the REFCOUNT IS at 2
```

Παραδειγμα 2

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 10 7 37
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        10
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Sun Apr  2 23:35:16 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1645] 02 Apr 23:35:16.065 . file.c:200 Creating directory structure: testdb/si
```

.....

```
[1645] 02 Apr 23:35:16.071 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0
0 searching key-0hed 0 ops
0 searching key-0hed 0 ops
[1645] 02 Apr 23:35:16.077 . db.c:36 Closing database 0
```

.....

```
[1645] 02 Apr 23:35:16.079 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 adding key-0
0 adding key-0inished 0 ops
0 adding key-0inished 0 ops
0 adding key-0inished 0 ops
0 adding key-0inished 0 ops
0 adding key-0inished 0 ops
0 adding key-0inished 0 ops
[1645] 02 Apr 23:35:16.085 . db.c:36 Closing database 7
[1645] 02 Apr 23:35:16.085 . sst.c:595 IN sst_merge the REFCOUNT IS at 2
```

Παραδειγμα 3

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 47 3 27
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        47
IndexSize:      0.0 MB (estimated)
DataSize:       0.0 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Sun Apr  2 23:37:57 2023
CPU:            1 *  Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
CPUCache:       6144 KB

[1663] 02 Apr 23:37:57.279 . file.c:200 Creating directory structure: testdb/si
```

.....

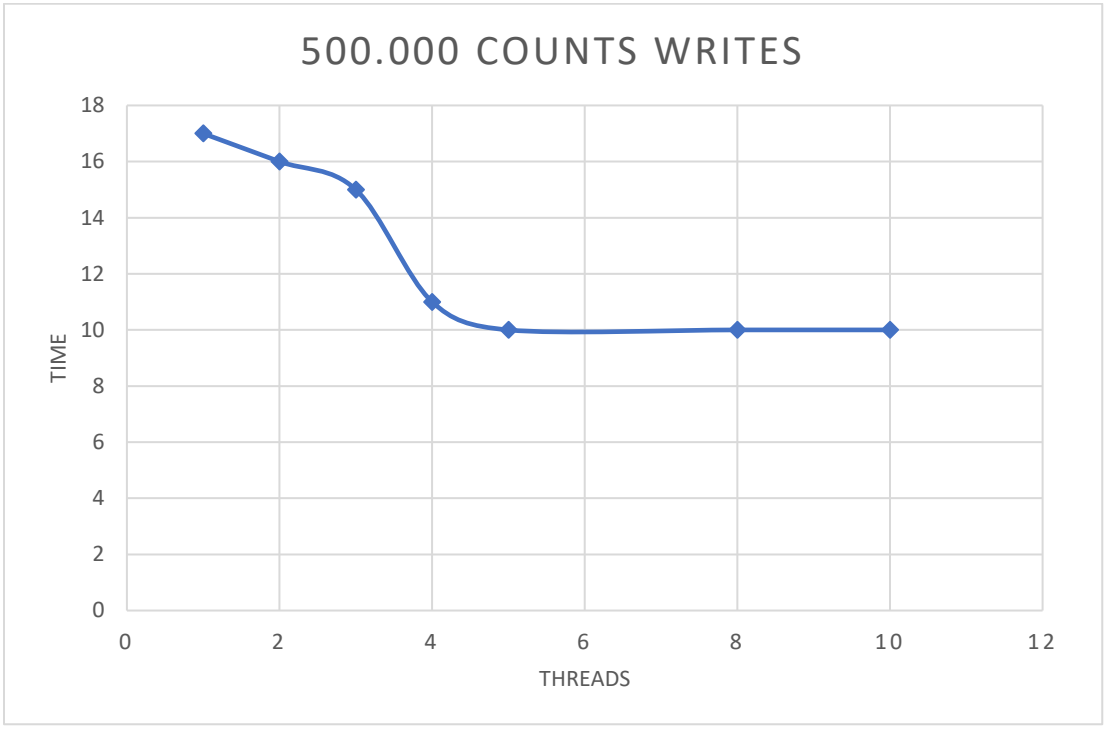
```
[1663] 02 Apr 23:37:57.284 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
[1663] 02 Apr 23:37:57.289 . db.c:36 Closing database 0
```

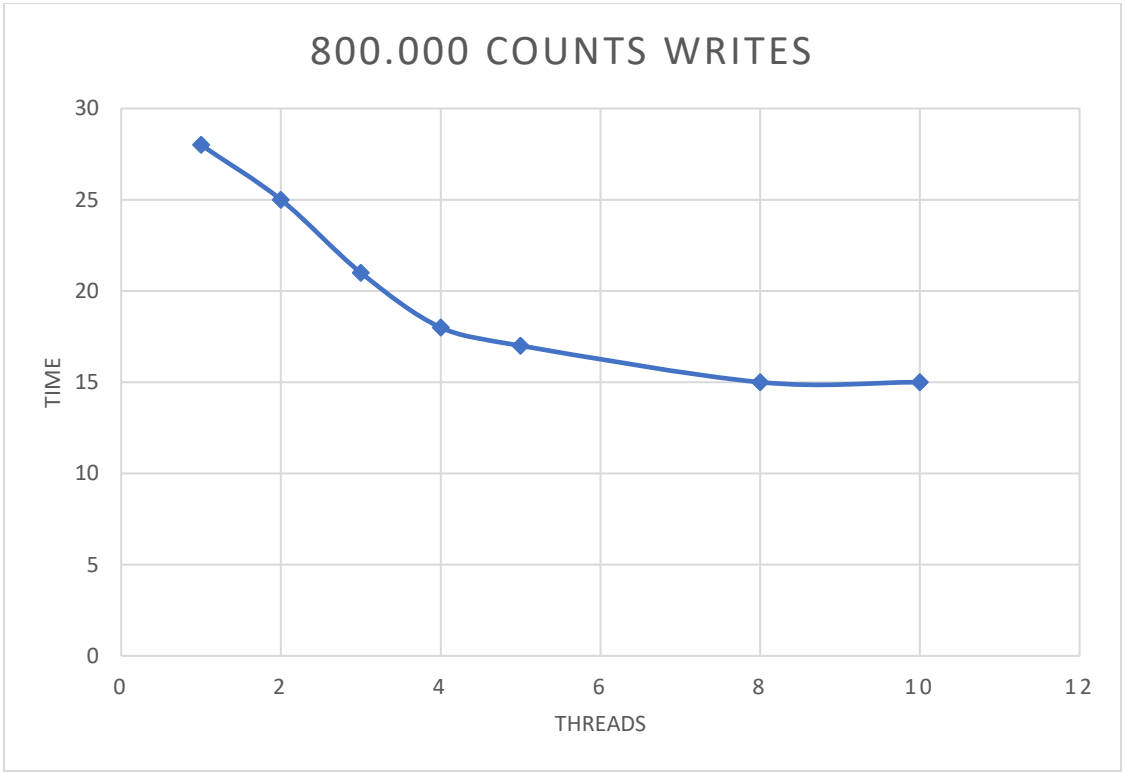
.....

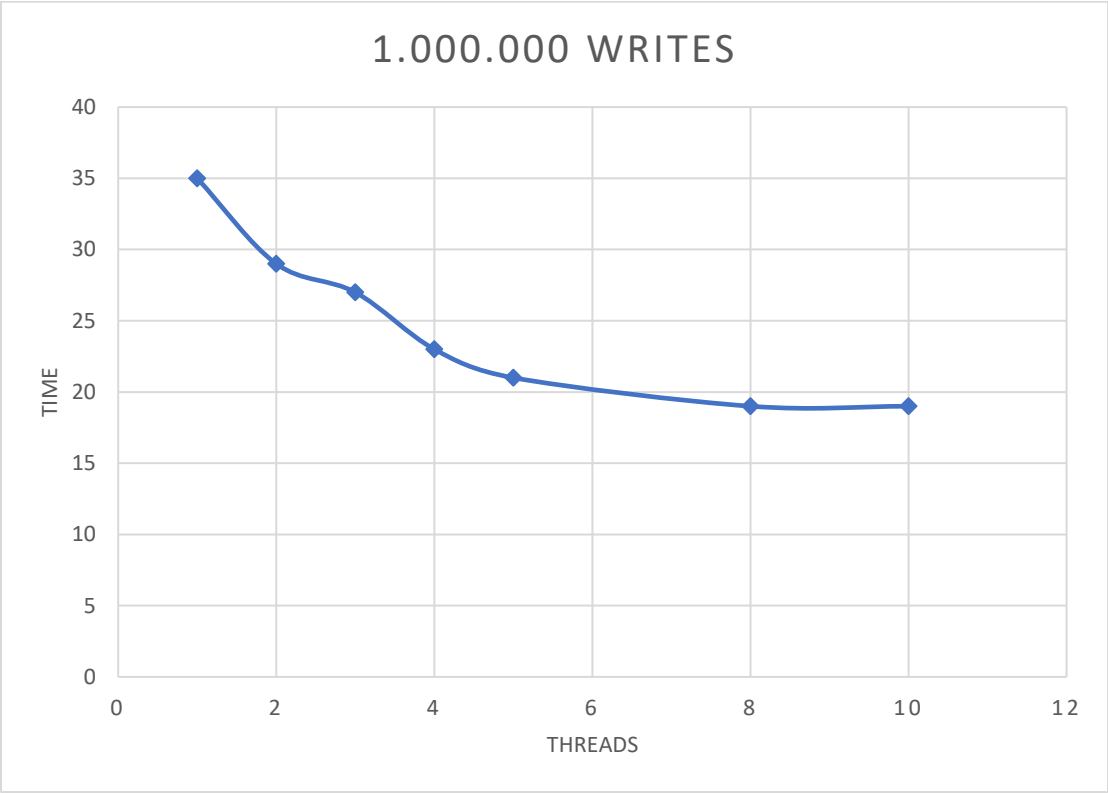
```
[1663] 02 Apr 23:37:57.292 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10
11 adding key-11
0 adding key-0
1 adding key-1inished 0 ops
2 adding key-2
3 adding key-3
4 adding key-4
5 adding key-5
6 adding key-6
7 adding key-7
8 adding key-8
9 adding key-9
10 adding key-10
11 adding key-11
[1663] 02 Apr 23:37:57.295 . db.c:36 Closing database 35
```

Στατιστικά της απόδοσης των λειτουργιών:

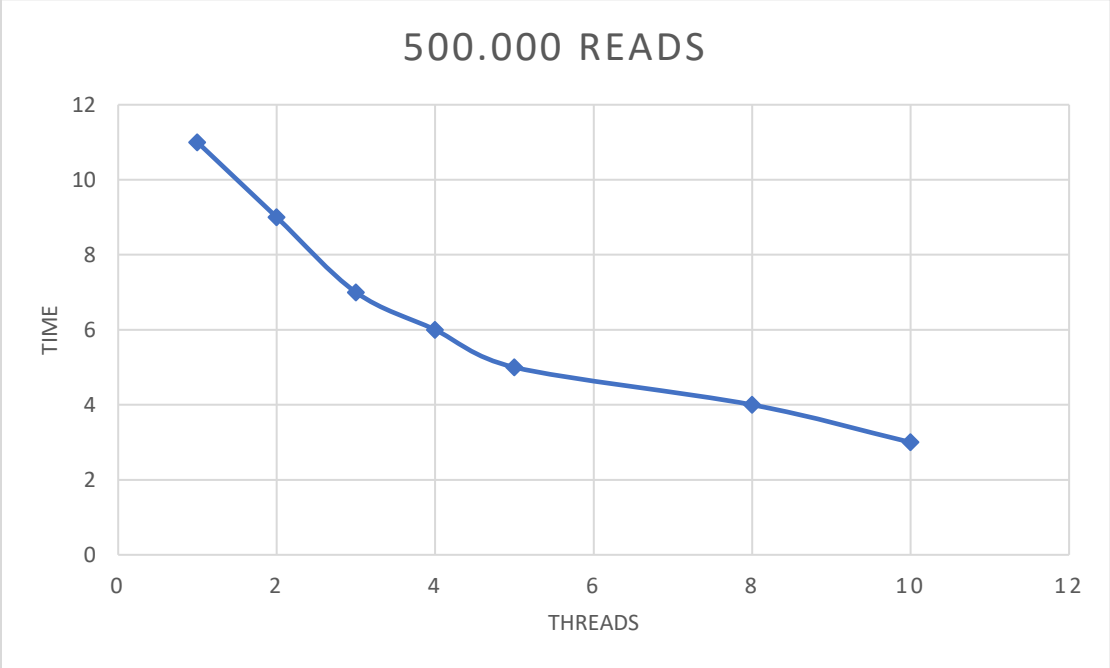
Writes

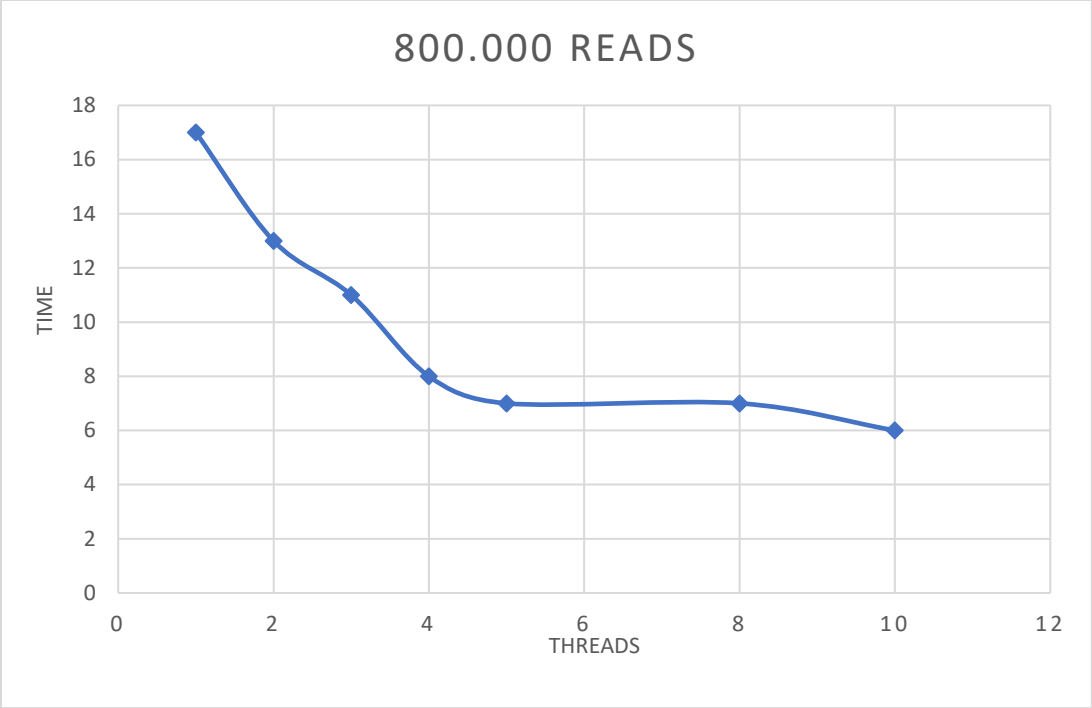


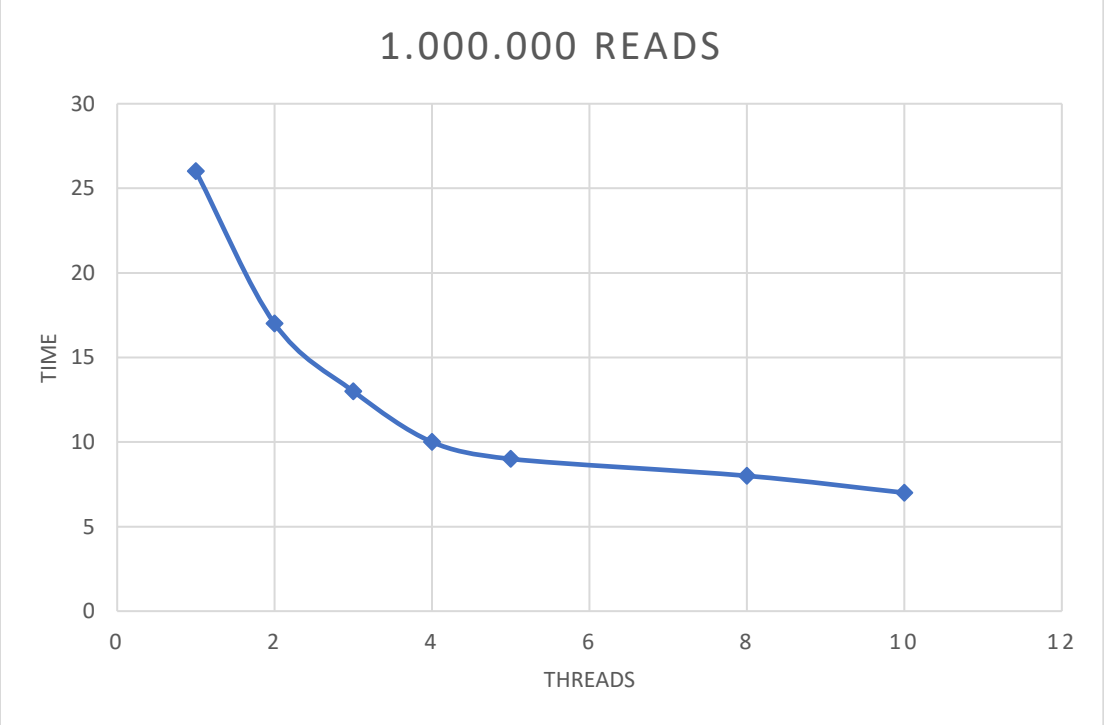



1.000.000 WRITES
TIME
THREADS
0
5
10
15
20
25
30
35
40
0
2
4
6
8
10
12

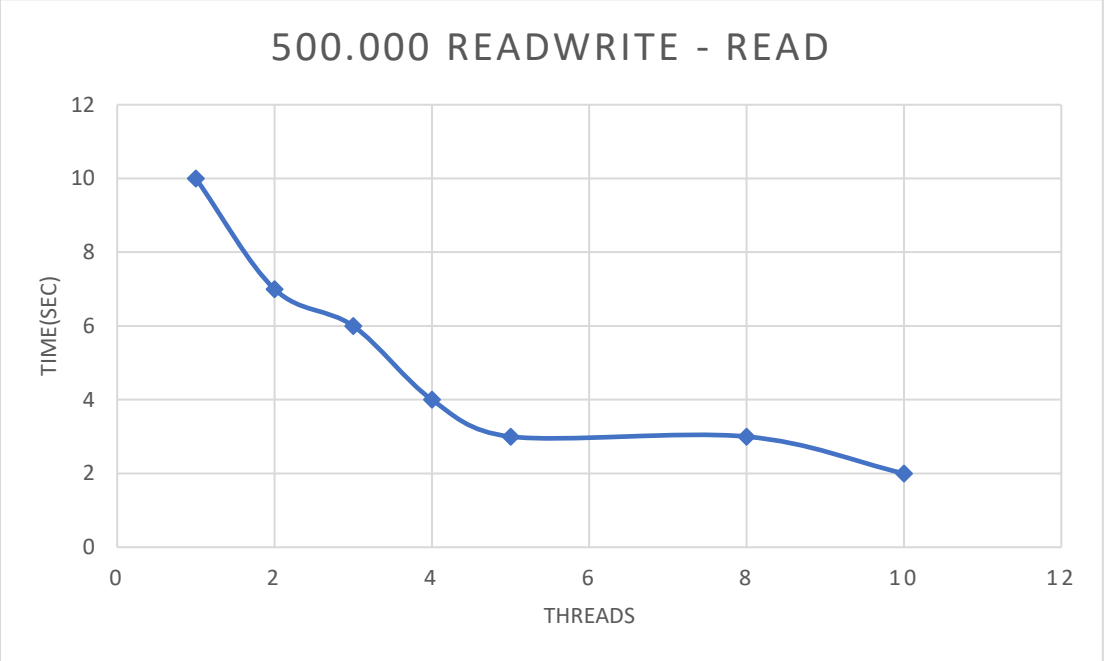
Reads

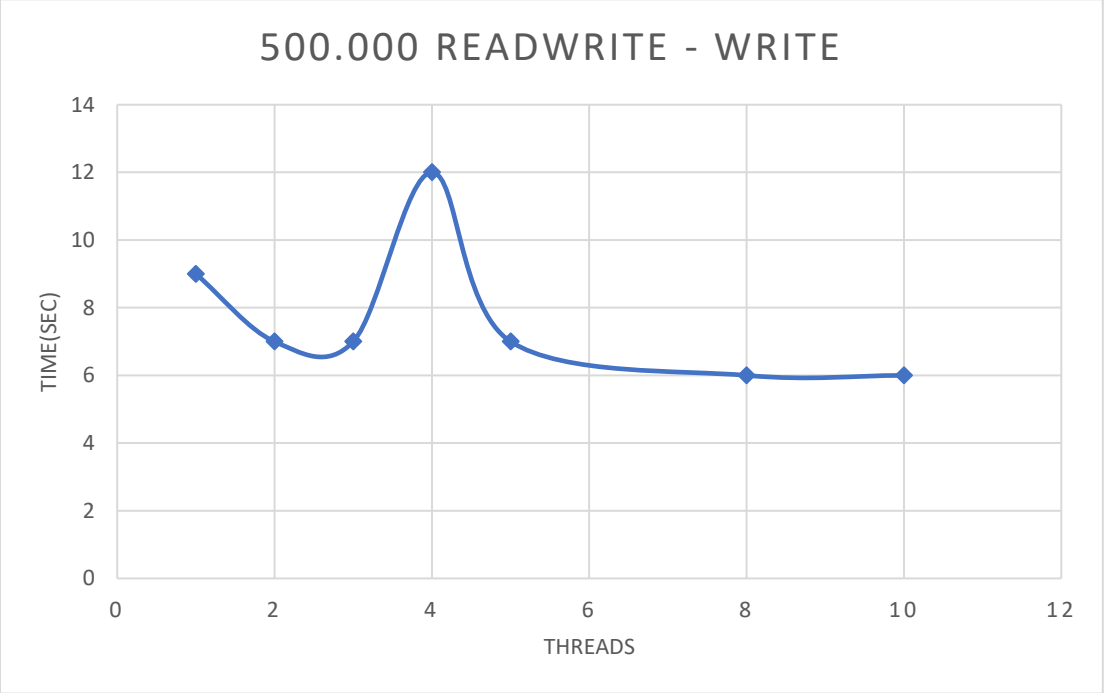

500.000 READS
TIME
THREADS
0
2
4
6
8
10
12
0
2
4
6
8
10
12

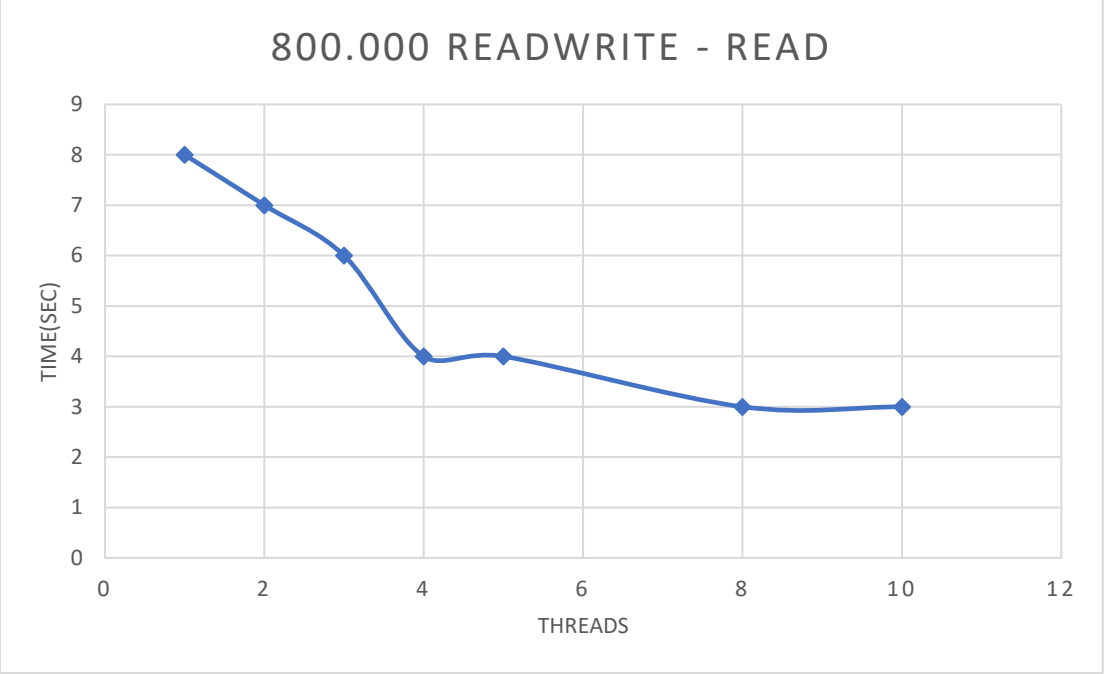

800.000 READS
TIME
THREADS
0
2
4
6
8
10
12
14
16
18

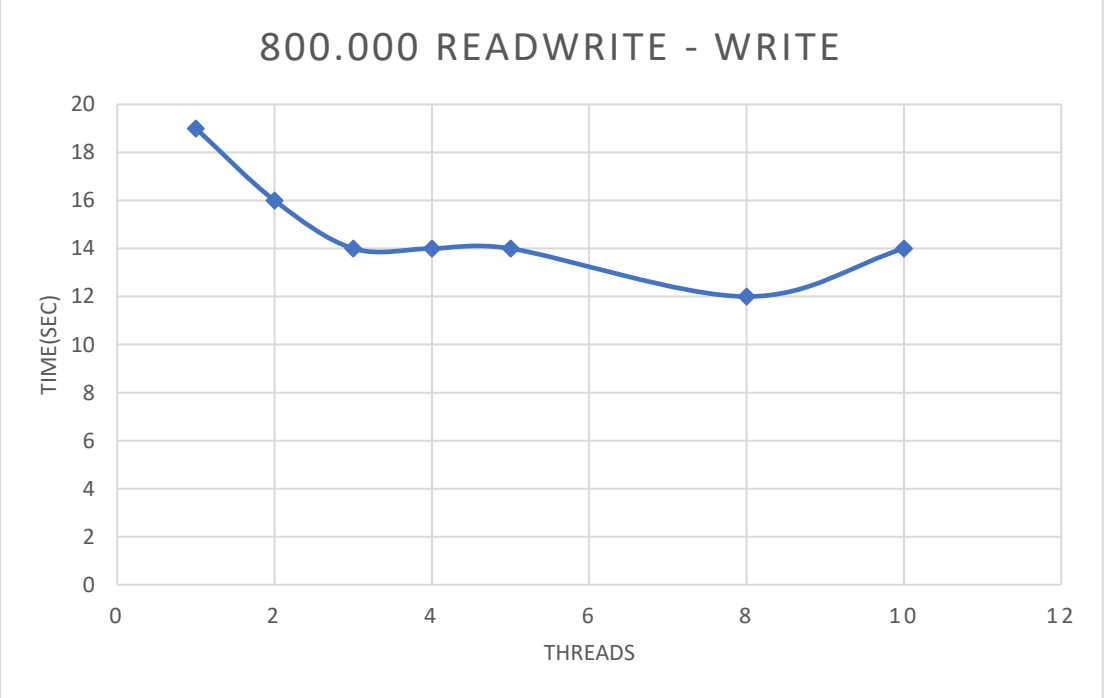

1.000.000 READS
TIME
THREADS
0
5
10
15
20
25
30
0
2
4
6
8
10
12

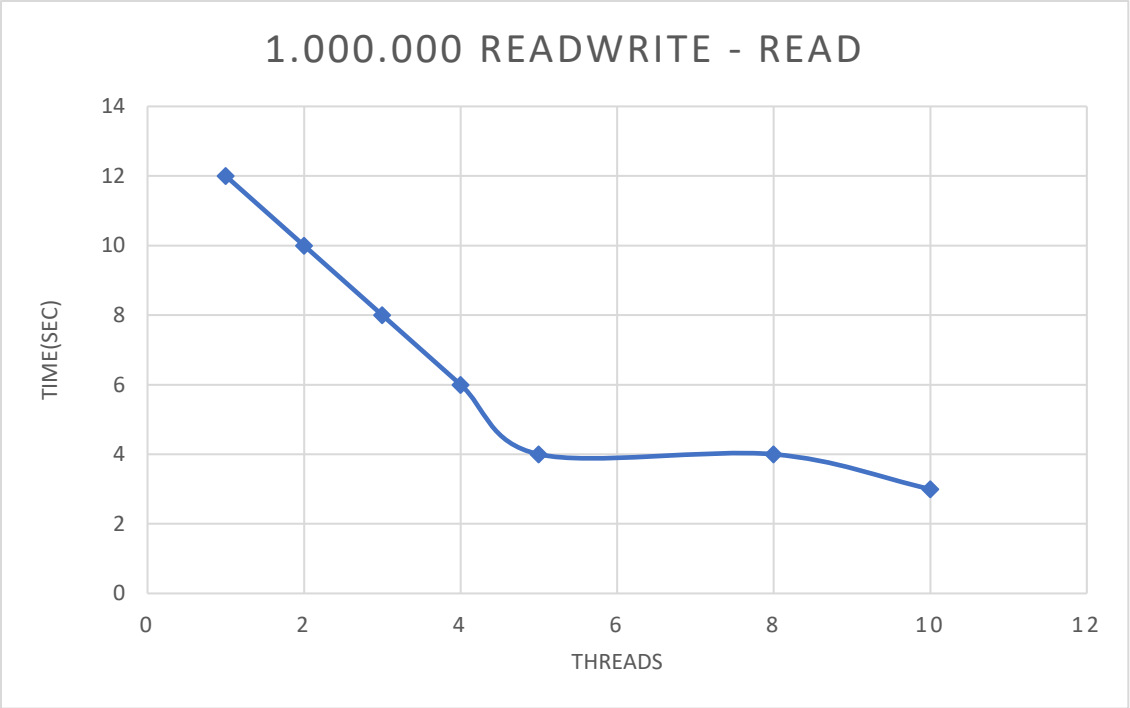
ReadWrite για 50%

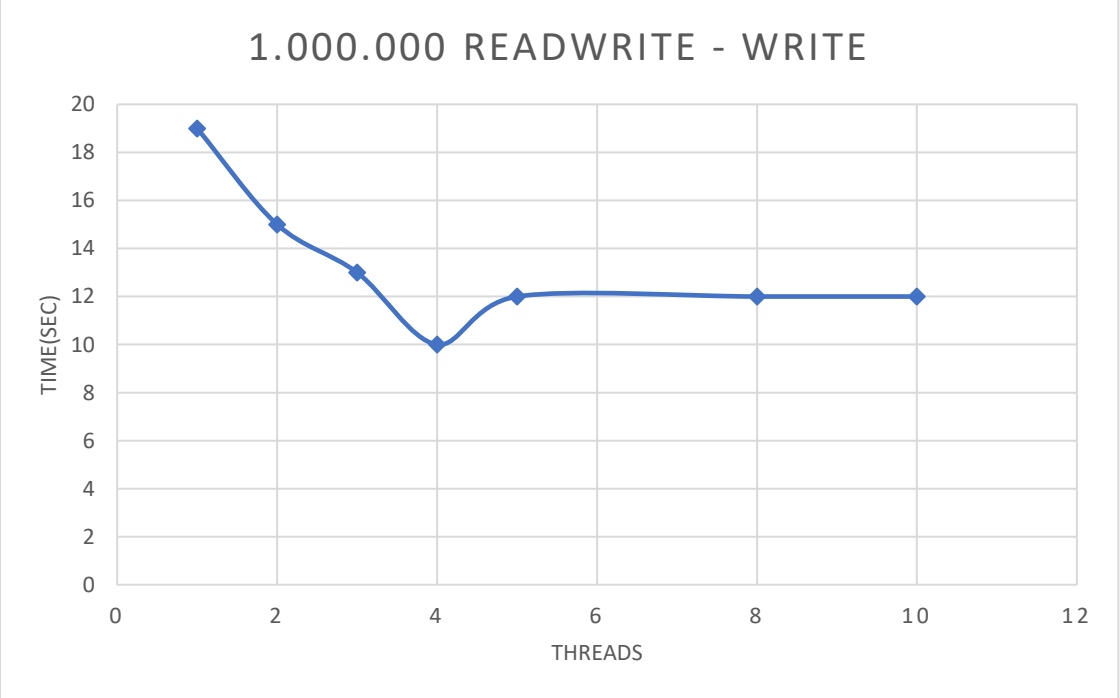





800.000 READWRITE - READ
TIME(SEC)
0
1
2
3
4
5
6
7
8
9
0
2
4
6
8
10
12
THREADS


800.000 READWRITE - WRITE
TIME(SEC)
0
2
4
6
8
10
12
14
16
18
20
0
2
4
6
8
10
12
THREADS


1.000.000 READWRITE - READ
TIME(SEC)
THREADS
0
2
4
6
8
10
12
14


1.000.000 READWRITE - WRITE
TIME(SEC)
THREADS
0
2
4
6
8
10
12
14
16
18
20

Σύντομη περιγραφή τρίτου βήματος ταυτοχρονισμού με χρήση διαφορετικών κλειδαριών για το memtable και τα αρχεία sst:

Στο 3ο βήμα μας ζητείται η δημιουργία διαφορετικών κλειδαριών για το memtable και τα αρχεία sst, έτσι ώστε διαφορετικά νήματα να λειτουργούν ταυτόχρονα εφόσον δεν τροποποιούν το ίδιο μέρος του συστήματος το ίδιο χρονικό διάστημα. Για να το πετύχουμε αυτό, εφαρμόσαμε δύο φορές τον αλγόριθμο γραφέων-αναγνωστών , μία στο memtable.c – στις συναρτήσεις memtable_add και memtable_get – και μία στο sst.c – στις συναρτήσεις sst_merge και sst_get. Η λογική που σκεφτήκαμε είναι να τρέχουν τα νήματα παράλληλα στο memtable και στο sst, δυσκολευτήκαμε στο αρχείο sst γιατί οι αλλαγές που κάναμε μάλλον επηρεάζουν το ήδη υπάρχον νήμα.
Η συγκεκριμένη υλοποίηση ενώ κάνει compile και δουλεύει για μικρές τιμές του count, σε μεγαλύτερες τιμές ή σε πιο πολλά threads σμας οδηγεί σε segmentation fault.
Έτσι αποφασίσαμε να στείλουμε τα αρχεία του 2ου βήματος , και να αναφέρουμε τις αλλαγές που κάναμε στα memtable.c , sst.c μόνο στην αναφορά μας.

Παρατίθονται τα κομμάτια κώδικα:

Memtable.c

```
……..
int memtable_add(MemTable* self, const Variant* key, const Variant* value)
{
    int ret_val;
    pthread_mutex_lock(&mem_lock);                  // readers – writer
algorithm
    while (act_mem_get + act_mem_add > 0){
        wait_mem_add++;                             //
        pthread_cond_wait(&writer, &mem_lock);      //
        wait_mem_add--;                             //
    }
    act_mem_add++;                                  //
    pthread_mutex_unlock(&mem_lock);
    ret_val = _memtable_edit(self, key, value, ADD);
    act_mem_add--;
    pthread_mutex_lock(&mem_lock);

    if (wait_mem_get > 0 && act_mem_add == 0){
        pthread_cond_broadcast(&memreader);     // signal memtable reader
    }
    else if (wait_mem_add > 0 && act_mem_add == 0){
        pthread_cond_signal(&writer);           // signal memtable_add
    }
    pthread_mutex_unlock(&mem_lock);
    return ret_val;
}
…….
```

```
……
int memtable_get(SkipList* list, const Variant *key, Variant* value)
{
    pthread_mutex_lock(&mem_lock);  // reader – writers algorithm
    while (act_mem_add > 0){                                    //
        wait_mem_get++;                                         //
        pthread_cond_wait(&memreader, &mem_lock);               //
        wait_mem_get--;                                         //
    }
    act_mem_get++;
    pthread_mutex_unlock(&mem_lock);
    SkipNode* node = skiplist_lookup(list, key->mem, key->length);

    if (!node){
        act_mem_get--;
        pthread_mutex_lock(&mem_lock);
        if (act_mem_get == 0 && wait_mem_add > 0){
            pthread_cond_signal(&writer);
        }

        pthread_mutex_unlock(&mem_lock);
        return 0;
    }
……
```

Sst.c

```
……
void sst_merge(SST* self, MemTable* mem)
#ifdef BACKGROUND_MERGE
{
     pthread_mutex_lock(&sst_lock);

     while (act_sst_get + act_sst_merge > 0){  // reader-writer algorithm
        wait_sst_merge++;
        pthread_cond_wait(&merge, &sst_lock);
        wait_sst_merge--;
    }
    pthread_mutex_unlock(&sst_lock);


   pthread_mutex_lock(&self->cv_lock);

……
```

```
……
    pthread_mutex_unlock(&self->immutable_lock);


    self->merge_state |= MERGE_STATUS_INPUT;

    pthread_cond_signal(&self->cv);
    pthread_mutex_unlock(&self->cv_lock);

    pthread_mutex_lock(&sst_lock);                          //
    act_sst_get--;                                          //
    if (act_sst_merge == 0 && wait_sst_get > 0){
        pthread_cond_broadcast(&sstreader);       // broadcast sst_get
    }
    pthread_mutex_unlock(&sst_lock);
}
…..
```

```
…….
int sst_get(SST* self, Variant* key, Variant* value)
{
#ifdef BACKGROUND_MERGE

    pthread_mutex_lock(&sst_lock); // reader-writerlgorithm
    while (act_sst_merge > 0){
        wait_sst_get++;
        pthread_cond_wait(&sstreader, &sst_lock);
        wait_sst_get--;
    }
    pthread_mutex_unlock(&sst_lock);
    act_sst_get++;

    int ret = 0;

    pthread_mutex_lock(&self->cv_lock);
    if (self->immutable)
    {
        DEBUG("Serving sst_get request from immutable memtable");
        ret = memtable_get(self->immutable_list, key, value);
    }
    pthread_mutex_unlock(&self->cv_lock);

    if (ret){
        act_sst_get--;
        pthread_mutex_lock(&sst_lock);
```

```
        if (act_sst_get == 0 && wait_sst_merge > 0){
             pthread_cond_signal(&merge);        // signal sst_merge
        }
        pthread_mutex_unlock(&sst_lock);
        return ret;
    }
    pthread_mutex_lock(&self->lock);
#endif
……
……
            act_sst_get--;
            pthread_mutex_lock(&sst_lock);
            if (act_sst_get == 0 &&  wait_sst_merge > 0){
                pthread_cond_signal(&merge);     // signal sst_merge
            }
            pthread_mutex_unlock(&sst_lock);
            return opt == ADD;
        }
    }

#ifdef BACKGROUND_MERGE
    pthread_mutex_unlock(&self->lock);
#endif

    act_sst_get--;
    pthread_mutex_lock(&sst_lock);
    if (wait_sst_merge > 0){
        pthread_cond_signal(&merge);            // signal sst_merge
    }
    pthread_mutex_unlock(&sst_lock);

    return 0;
}
……..
```