Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

Δημητριάδης Γεώργιος - ΑΜ: 5209
Δημητρίου Αριστοτέλης - ΑΜ: 5211
Τζιάσιος Κίμων - ΑΜ: 5365

ΜΥΥ601 Λειτουργικά Συστήματα
Εαρινό Εξάμηνο 2024

Πίνακας περιεχομένων

Εισαγωγή

Στην πρώτη εργαστηριακή άσκηση, μας δίνεται μια μηχανή αναζήτησης βασισμένη στο δέντρο LSM η οποία είναι μία απλή μηχανή που αποθηκεύει τα δεδομένα ως ένα ζεύγος κλειδιού-τιμής.

Περιήγηση στις λειτουργίες

Υπάρχουσα μηχανή

Οι βασικές λειτουργίες είναι η add που δέχεται ως παράμετρο ένα ζεύγος κλειδιού-τιμής και η get που δέχεται ως παράμετρο ένα κλειδί και αναζητά την αντίστοιχη τιμή εφόσον υπάρχει αποθηκευμένη στη δομή. Λεπτομέρειες για το πώς η δομή διατηρεί τα αποθηκευμένα δεδομένα στην μνήμη ή στο δίσκο και με ποιόν τρόπο μετακινούνται από το ένα στο άλλο αναγράφονται στην εκφώνηση της άσκησης. Το βασικό κομμάτι στη μηχανή είναι όταν το memtable μετακινείται στον δίσκο συγχωνεύεται με τα αρχεία των οποίων τα κλειδιά επικαλύπτονται με τα κλειδιά του memtable. Μετέπειτα, ενεργοποιείται σύμπτυξη με βάση κάποια προκαθορισμένα κατώφλια.

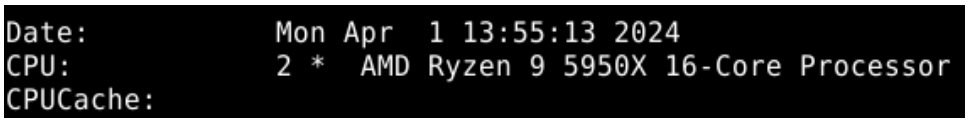
Στην αρχική υλοποίηση υπάρχουν *δύο νήματα,* ένα στη μηχανή και ένα στην εφαρμογή. Το πρώτο νήμα είναι υπεύθυνο για τις add-get (μία μετά την άλλη) ενώ το δεύτερο για την σύμπτυξη(compaction), το οποίο δημιουργείται και αρχικοποιείται στο sst.c και πιο συγκεκριμένα στην συνάρτηση sst_new.

Σύντομη περιήγηση:

- Στο αρχείο bench.c περιέχεται η main η οποία ανάλογα με τις παραμέτρους που ο χρήστης δίνει στο τερματικό, είτε ανάγνωσης είτε εγγραφής, καλεί τις αντίστοιχες συναρτήσεις οι οποίες υλοποιούνται στο kiwi.c
- Στο αρχείο kiwi.c υπάρχουν ήδη δύο σημαντικές συναρτήσεις η _read_test και η _write_test. Αυτές ανοίγουν το database, προσθέτουν χαρακτηριστικά ζεύγη(κλειδιά-τιμή) καλώντας από το db.c την db_add είτε διαβάζουν με την db_get. Στο τέλος τυπώνονται κάποια βασικά κόστη, όπως π.χ. second/operation.
- Στο αρχείο db.c οι βασικές συναρτήσεις είναι η db_add και db_get. Μέσα στη db_add γίνεται έλεγχος αν το memtable χρειάζεται compaction και μετά με τη σειρά της καλείται η sst_merge αλλιώς προσθέτουμε το ζεύγος στο memtable. Στη db_get αναζητούμε πρώτα στο memtable. Εάν δεν βρεθεί η αναζήτηση γίνεται στο sst.

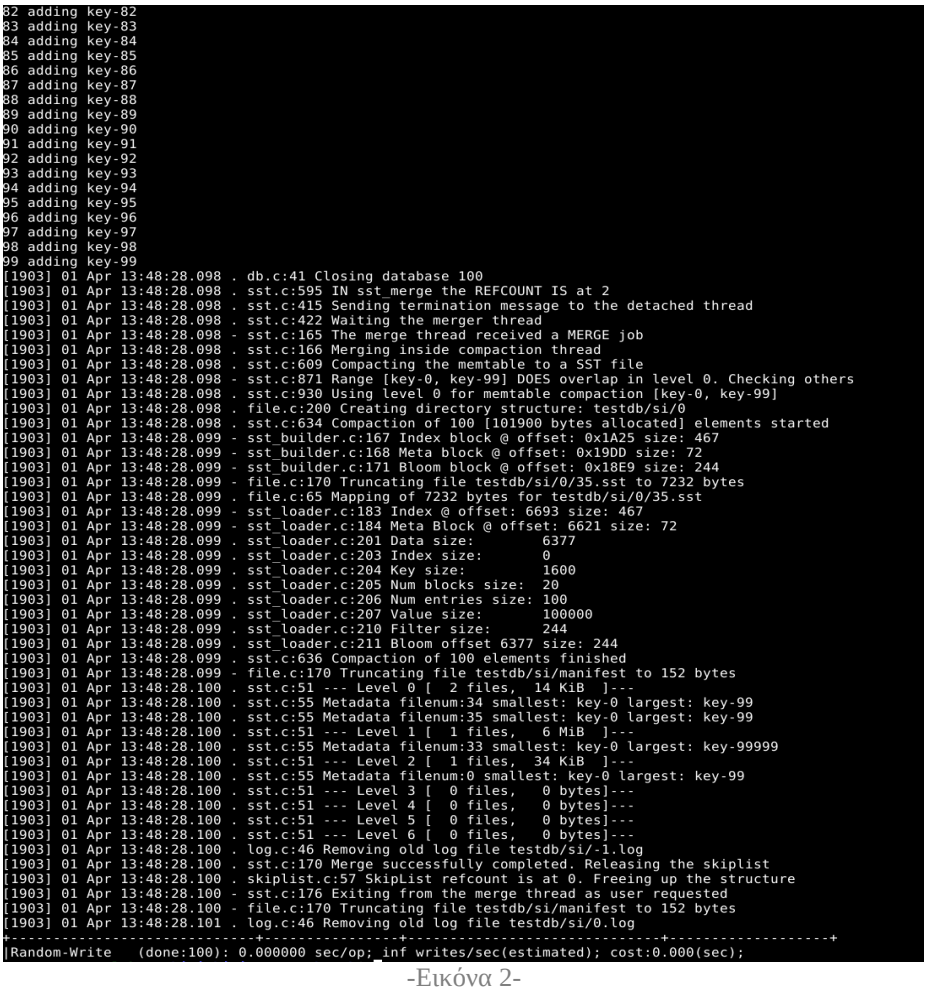
Αποτελέσματα βασικής λειτουργίας

Αρχικά το μηχάνημα με το οποίο τρέχουμε τους κώδικες είναι εξοπλισμένο με τον εξής επεξεργαστή.(εικόνα 1)



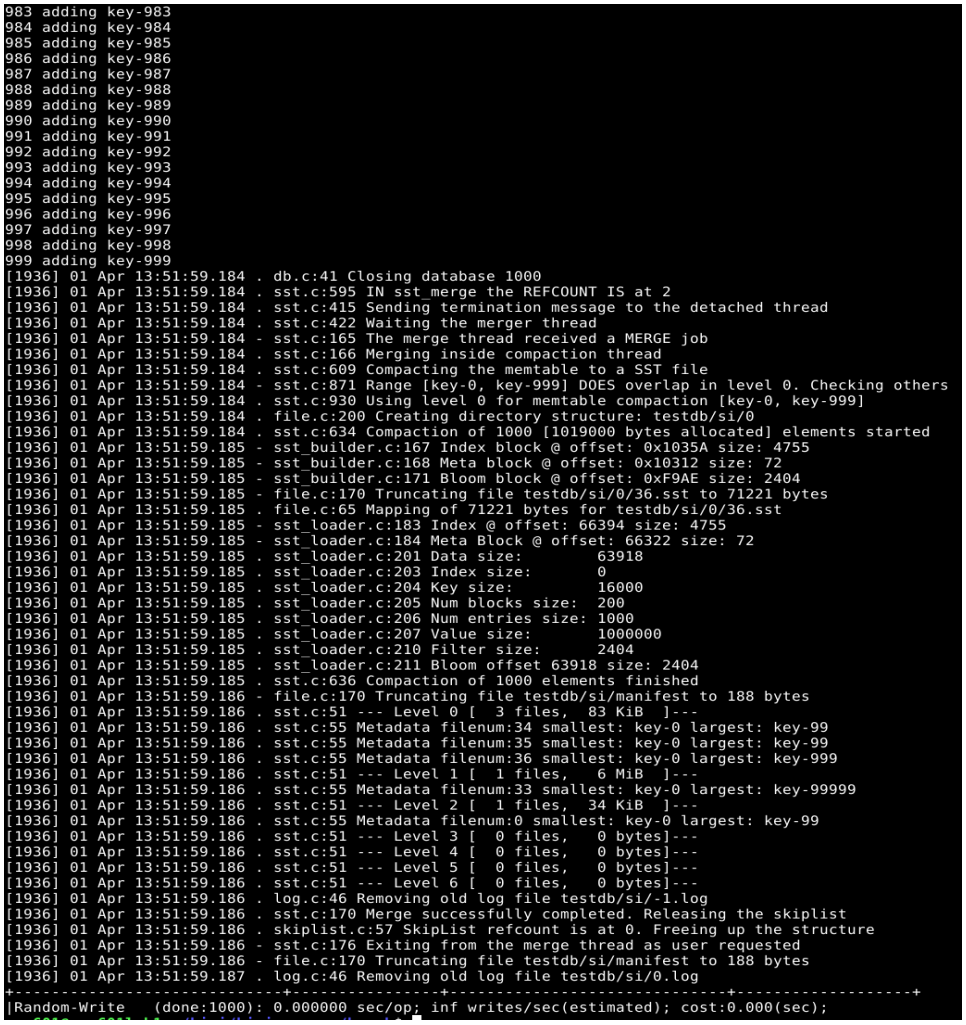
-Εικόνα 1-

Για αρχή θα ελέγξουμε τι γίνεται για 100 ζεύγη όταν εκτελούμε από το τερματικό ./kiwi-bench write 100. (εικόνα 2)



-Εικόνα 2-

Στη συνέχεια εκτελούμε το read με την εντολή ./kiwi-bench read 100 και όπως είναι εμφανές (στην εικόνα 3) προκύπτει:



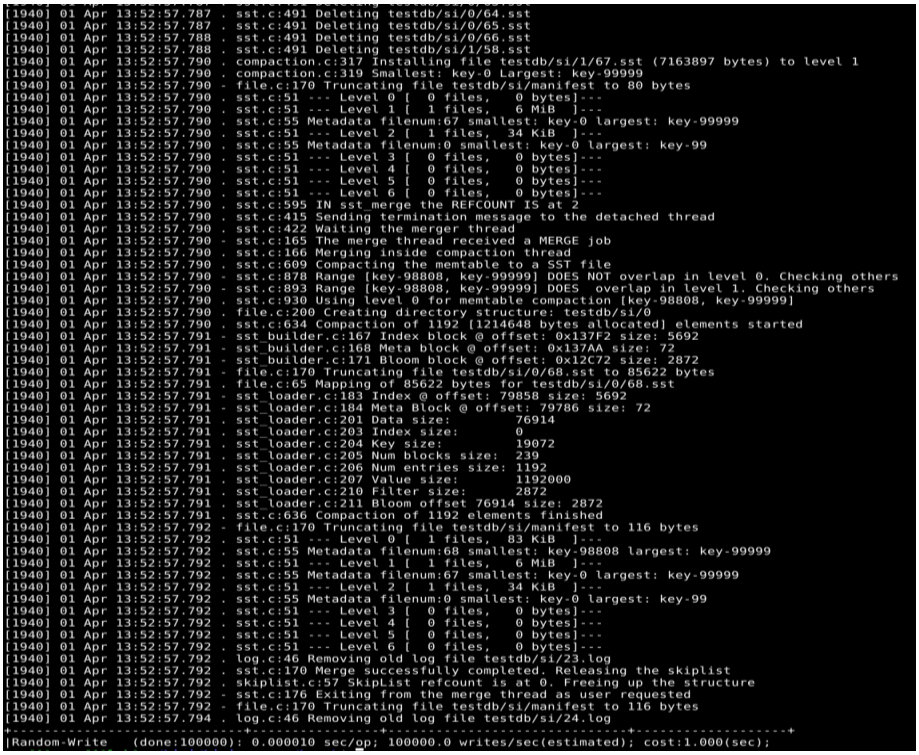
-Εικόνα 3-

Όπως παρατηρούμε επειδή τα ζεύγη είναι λίγα η χρονομέτρηση μας επιστρέφει μηδενικά αποτελέσματα.

Εκτελούμε την ίδια διαδικασία για 1.000, 10.000, 100.000 και 1.000.000 κλειδιά αντίστοιχα και αρχίζουμε να παρατηρούμε, όπως φαίνεται παρακάτω αισθητή διαφορά σε τιμές πάνω από 100.000.

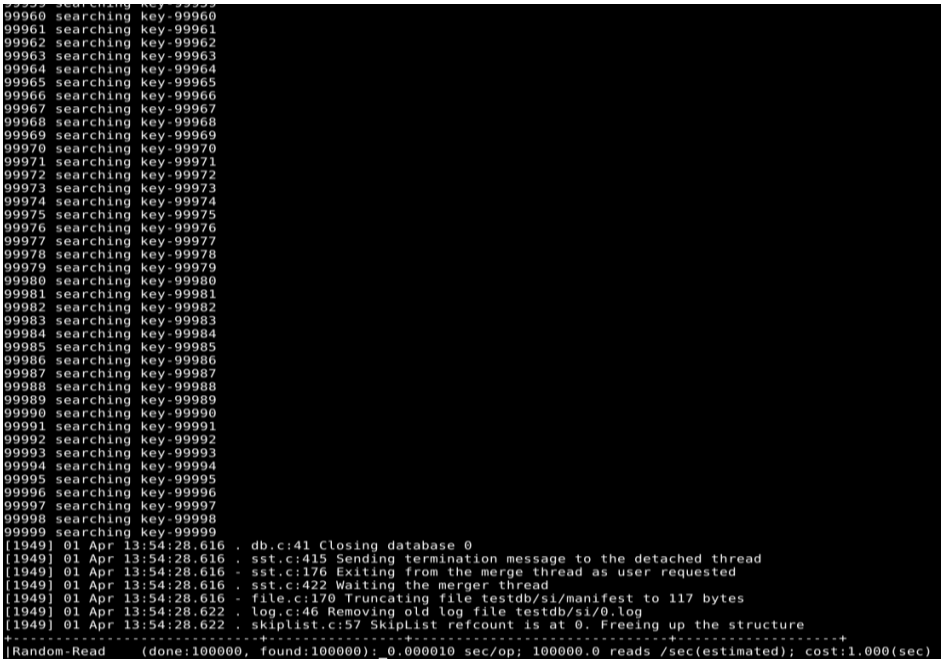
Πιο συγκεκριμένα:

Για 100.000 ζεύγη εκτελούμε ./kiwi-bench write 100000 και μας εμφανίζεται στην οθόνη (εικόνα 4):



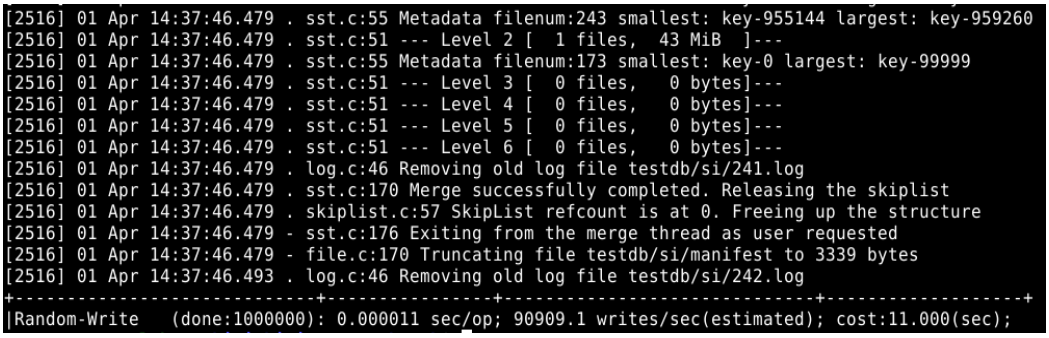
-Εικόνα 4-

Αντίστοιχα εκτελούμε το read(εικόνα 5) με την εντολή ./kiwi-bench read 100000.



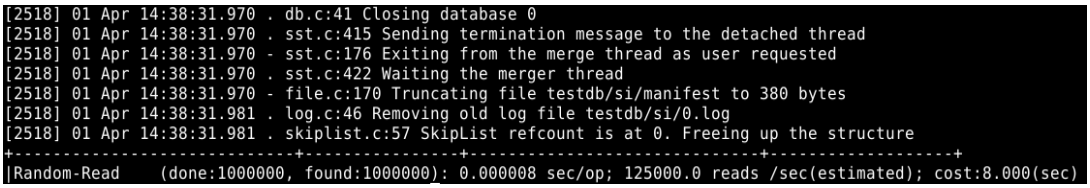
-Εικόνα 5-

Για 1.000.000 ζεύγη: ./kiwi-bench write 1000000. (εικόνα 6)



-Εικόνα 6-

Τέλος, εκτελούμε στο τερματικό το read με την εντολή ./kiwi-bench read 1000000. (εικόνα 7)



-Εικόνα 7-

Πλέον είναι αισθητό το κόστος ανάγνωσης και εγγραφής καθώς για 1.000.000 εγγραφές χρειαζόμαστε 11 δευτερόλεπτα και για 1.000.000 αναγνώσεις 8 δευτερόλεπτα.

Πολυνηματική Υλοποίηση: Για την πολυνηματική υλοποίηση μας ζητείται να επιλύσουμε το πρόβλημα ταυτοχρονισμού με αμοιβαίο αποκλεισμό στην εκτέλεση των add και get λειτουργιών. Εμείς προσπαθήσαμε να λύσουμε το πρόβλημα με εφαρμογή συγχρονισμού αναγνώστη-γραφέων.

Τροποποιήσεις στη μηχανή:

Αλλαγές στον ήδη υπάρχων κώδικα:

- Στο bench.h υλοποιούμε το ακόλουθο struct(εικόνα 8).

```
//ftiaxoume ena kainourgio thread panw sto arxeio h wste na mas dieukolinei kai na einai pio katharos o kwdikas.
typedef struct {
    int r;
    int tnumber;
        int count;
        int threads;
}args;
```

-Εικόνα 8-

Οι μεταβλητές count και r προϋπάρχουν, εμείς προσθέτουμε δύο ακόμη μεταβλητές. Η πρώτη είναι η tnumber, για να παρακολουθούμε κάθε thread. Η δεύτερη είναι η threads, η οποία κρατάει τον αριθμό από τα νήματα που εισάγει ο χρήστης στη γραμμή εντολών

- Στο bench.c ορίζουμε την καινούρια λειτουργία rw μέσα στη main.

```
else if (strcmp(argv[1], "rw") == 0) {
     int r = 0;
     count = atoi(argv[2]);
     if (argc == 6){                                // Profanws allaksame  thn synthiki giati pleon tha vazoume parapanw orismata gia to readwrite (threads kai pososto)
            r = 1;
     }
     if(argc < 5){
            fprintf(stderr,"Usage: db-bench <rw> <count> <number of threads> <percentage(%)>\n");
            return 1;
     }
     int threads = atoi(argv[3]);
     int perc= atoi(argv[4]);                       // apothikevoume to pososto kai threads
     _readwrite(count, r,threads, perc);      //pername ta orismata sthn synarthsh pou tha kanei read kai write
```

-Εικόνα 9-

Όπως είναι εμφανές στην εικόνα 9, πρώτα ελέγχεται εάν η πρώτη παράμετρος της γραμμής εντολών είναι "rw". Αυτό γίνεται με τη χρήση της συνάρτησης strcmp που συγκρίνει δύο αλφαριθμητικά. Αν η σύγκριση επιστρέψει 0, τότε τα αλφαριθμητικά είναι ίδια και εκτελείται η συνθήκη για τη λειτουργία "rw". Έπειτα, έλεγχος γίνεται για τον αριθμό των ορισμάτων. Αν ο αριθμός των ορισμάτων είναι λιγότερος από 5, τότε εκτυπώνεται ένα μήνυμα λάθους που ενημερώνει το χρήστη για τη σωστή χρήση του προγράμματος και το πρόγραμμα τερματίζεται με επιστροφή τιμής 1. Αν ο αριθμός των ορισμάτων είναι τουλάχιστον 5, τα ορίσματα count, threads και perc αντιστοιχίζονται από τη γραμμή εντολών σε ακέραιες μεταβλητές. Η συνάρτηση _readwrite καλείται με τα ορίσματα count, r, threads και perc, όπου θα εκτελέσει την ανάγνωση και εγγραφή στη βάση δεδομένων.

- Στο kiwi.c

  - Στην αρχή του κώδικα ορίζουμε global το database ώστε στις παρακάτω συναρτήσεις να είναι ορατό.(εικόνα 10)

```
DB* db_for_threads;
```

-Εικόνα 10-

  - Έπειτα συντάξαμε την συνάρτηση _readwrite ως εξής.(εικόνα 11)

```
void _readwrite(int count, int r, int threads, int perc)
{
     int found;

    int i;
    double cost;
    long long start, end;
    pthread_t write_threads[100];
    pthread_t read_threads[100];
    int write_thread = threads * perc / 100; //apothikevoume to pososto twn nhmatwn gia write
    int read_thread = threads-write_thread;// apothikevoume to pososto twn nhmatwn gia read me mia aplh afairesh

    db_for_threads = db_open(DATAS);//Open thn global
    start = get_ustime_sec();

    for (i = 0; i < write_thread; i++)
    {
        args *bench_struct = malloc(sizeof(args));//ftiaxnoyme ena neo struct args me malloc kai antistoixoume tis times tou
        bench_struct->count = ((count * perc) / 100);
        bench_struct->r = r;
        bench_struct->threads = write_thread;
        bench_struct->tnumber = i;
        pthread_create(&write_threads[i], NULL,writer, (void *)bench_struct);//ftiaxoume ta threads kai pername ta orismata katallhla
    }
    for (i = 0; i < read_thread; i++)
    {
        args *bench_struct = malloc(sizeof(args));//ksana h idia diadikasia me panw...
        bench_struct->r = r;
        bench_struct->count = ((count * (100 - perc)) / 100);
        bench_struct->threads = read_thread;
        bench_struct->tnumber = i;
        pthread_create(&read_threads[i], NULL, reader , (void *)bench_struct);
    }

    for (i = 0; i < write_thread; i++)//kanoume join gia na perimenoun thn oloklhrwsh olwn twn allwn nhmatwn
    {
        pthread_join(write_threads[i], NULL);
    }
    for (i = 0; i < read_thread; i++)
    {
        void*result;
        pthread_join(read_threads[i],&result);//edw vazoume result etsi wste na metrame sto found ta kleidia pou vrethikan
        found += (int)result;

    }

    db_close(db_for_threads);

    end = get_ustime_sec();
    cost = end - start;
    printf(LINE);
    printf("|Threaded-ReadWrite (done:%d): %.6f (found:%d) sec/op; %.1f ops/sec(estimated); cost:%.3f(sec);\n"
        , count, (double)(cost / count * threads)
        ,found, (double)(count * threads / cost), (double)cost);
}
```

-Εικόνα 11-

Αρχικά, δηλώνονται μεταβλητές, όπως η found που θα χρησιμοποιηθεί για την αποθήκευση του αριθμού των κλειδιών που βρέθηκαν κατά τη διάρκεια των αναζητήσεων. Έπειτα, υπάρχει μια διαδικασία δημιουργίας νημάτων για εγγραφή και ανάγνωση δεδομένων στη βάση. Χρησιμοποιούνται δύο πίνακες write_threads και read_threads για την αποθήκευση των νημάτων που θα δημιουργηθούν. Ο αριθμός των νημάτων εγγραφής υπολογίζεται από την μεταβλητή write_thread, όπως επίσης και ο αριθμός των νημάτων ανάγνωσης read_thread. Στη συνέχεια, δημιουργούνται νήματα για εγγραφή και ανάγνωση δεδομένων. Για κάθε νήμα εγγραφής, δημιουργείται ένα νέο δομημένο bench_struct

με δεδομένα που αφορούν την εκάστοτε εργασία του νήματος. Αυτά τα δεδομένα περιλαμβάνουν τον αριθμό εγγραφών, τον αριθμό νημάτων, τον τρέχοντα αριθμό νήματος κ.λπ. Έπειτα, το κάθε νήμα δημιουργείται με τη συνάρτηση pthread_create. Ακολουθεί η αναμονή των νημάτων εγγραφής για ολοκλήρωση της εργασίας τους, χρησιμοποιώντας τη συνάρτηση pthread_join. Το ίδιο συμβαίνει και με τα νήματα ανάγνωσης. Κατόπιν, καλείται η db_close και υπολογίζεται ο χρόνος εκτέλεσης της διαδικασίας. Τέλος, εμφανίζεται ένα μήνυμα που περιέχει τα αποτελέσματα της διαδικασίας εγγραφής και ανάγνωσης, συμπεριλαμβανομένου του χρόνου εκτέλεσης και του αριθμού των εγγραφών που βρέθηκαν κατά τη διάρκεια της αναζήτησης. Επιπλέον επεξηγήσεις βρίσκονται μέσα στον κώδικα με την μορφή σχολίου.

Για τη διευκόλυνση της διαδικασίας δημιουργούμε δύο διαφορετικές συναρτήσεις, την reader και writer.

- Reader (εικόνα 12): είναι μια συνάρτηση που περνιέται ως pointer στα προηγούμενα pthread_create και εκτελεί τη δουλειά του αναγνώστη.

```
void *reader(void *arg)
{
    int i;
    int ret;
    long found = 0;
    char key[KSIZE + 1];
    Variant sk, sv;
    args *bench_struct = (args *)arg;//apo edw kai katw kanoume anathesh twn timwn ths domhs stis antistoixes metavlhtes kai orizoume first key last key kai thread keys pou tha mas xreiastoun parakatw
    int count = bench_struct->count;
    int tnumber = bench_struct->tnumber;
    int threads = bench_struct->threads;
    int r = bench_struct->r;
    int thread_keys = count / threads;
    int first_key = tnumber * thread_keys;
    int last_key = first_key + thread_keys - 1;
        if (tnumber == threads - 1) {
        last_key = count - 1;
    }
    for (i = first_key; i <= last_key; i++)
    {
        memset(key, 0, KSIZE + 1);
        if (r) {
            snprintf(key, KSIZE, "key-%d", rand() % count);
        } else {
            snprintf(key, KSIZE, "key-%d", i);
        }
    sk.length = KSIZE;
    sk.mem = key;
    ret = db_get(db_for_threads, &sk, &sv); //edw pernane thn global domh mas
    if (ret)
    {
        found++;
    }
    else
    {
        INFO("not found key#%s", sk.mem);
    }
            if ((i % 1000000) == 0) {
                    fprintf(stderr,"random read finished %d ops%30s\r",
                                    i,
                                    "");

                    fflush(stderr);
            }
}

printf("%d Thread with id %lu finished and found %ld keys\n\n\n", tnumber, pthread_self(), found); //typwnoume ton arithmo twn kleidiwn pou vrethike gia kathe thread mazi me ta id tous sto struct kai sto systhma
return (void*)found;
```

-Εικόνα 12-

Το νήμα ανάγνωσης λαμβάνει ένα όρισμα arg, το οποίο είναι ένα δείκτης σε μια δομή args. Από αυτή τη δομή, αντλούνται πληροφορίες σχετικά με το πόσες εγγραφές να αναζητηθούν από τη βάση δεδομένων, το ποιο είναι το νήμα (thread) και άλλες σχετικές πληροφορίες. Υπολογίζονται οι αρχικές και τελικές τιμές των κλειδιών (keys) που αντιστοιχούν σε αυτό το συγκεκριμένο νήμα. Αυτό γίνεται με βάση τον αριθμό των νημάτων και τον συνολικό αριθμό εγγραφών. Έπειτα, το νήμα επιχειρεί να διαβάσει τις εγγραφές από τη βάση δεδομένων. Για

κάθε εγγραφή, δημιουργείται ένα κλειδί (key) με βάση το είδος της ανάγνωσης (r). Στη συνέχεια, γίνεται αναζήτηση του κλειδιού στη βάση δεδομένων με τη χρήση της συνάρτησης db_get. Αν το κλειδί βρεθεί στη βάση δεδομένων, αυξάνεται ο μετρητής found. Κατά τη διάρκεια της διαδικασίας, ανά τόσες εγγραφές εκτυπώνεται ένα μήνυμα για το πόσες αναγνώσεις έχουν ήδη γίνει. Στο τέλος το νήμα εκτυπώνει ένα μήνυμα που αναφέρει τον αριθμό των κλειδιών που βρέθηκαν κατά τη διάρκεια της αναζήτησης, συνοδευόμενο από το ID του νήματος που το εκτέλεσε. Η συνάρτηση επιστρέφει τον αριθμό των κλειδιών που βρέθηκαν στο τέλος της εκτέλεσής της, χρησιμοποιώντας (void*)found.

- Writer (εικόνα 13): είναι μία συνάρτηση που περνιέται ως pointer στα προηγούμενα pthread_create και εκτελεί τη δουλειά του γραφέα.

```
void *writer(void *arg)
{
    int i;
    char key[KSIZE + 1];
    char val[VSIZE + 1];
    Variant sk, sv;
    args *bench_struct = (args*)arg;//Idia diadikasia opws sto reader apo panw
    int r = bench_struct->r;
    int count =bench_struct->count;
    int threads = bench_struct->threads;
    int tnumber = bench_struct->tnumber;
    int thread_keys = count / threads;
    int first_key = tnumber * thread_keys;
    int last_key = first_key + thread_keys - 1;
    if (tnumber == threads - 1) {
        last_key = count - 1;
    }
    long int total_writes_thread = 0;

    for (i = first_key; i <= last_key; i++)
    {

        memset(key, 0, KSIZE + 1);
        memset(val, 0, VSIZE + 1);
        if (r) {
            _random_key(key, KSIZE);
            snprintf(val, VSIZE, "val-%d", rand() % count);
        } else {
            snprintf(key, KSIZE, "key-%d", i);
            snprintf(val, VSIZE, "val-%d", i);
        }
       sk.length = KSIZE;
       sk.mem = key;
       sv.length = VSIZE;
       sv.mem = val;
       db_add(db_for_threads, &sk, &sv); // opws kai panw pername to global db

              total_writes_thread+= 1;
   }

       if ((i % 100000) == 0) {
                      fprintf(stderr,"random write finished %d ops%30s\r",
                                     i,
                                     "");

                      fflush(stderr);
              }

   printf("%d Thread with id %lu finished and wrote %ld keys\n\n\n", tnumber, pthread_self(), total_writes_thread);
   return NULL;
```

-Εικόνα 13-

Αυτή η συνάρτηση αναμένει ένα όρισμα arg, το οποίο είναι ένας δείκτης σε δομή args. Από αυτή τη δομή, αντλούνται οι παράμετροι που θα χρησιμοποιηθούν για την εγγραφή στη βάση δεδομένων. Υπολογίζονται οι αρχικές και τελικές τιμές των κλειδιών που αντιστοιχούν σε αυτό το συγκεκριμένο νήμα, ανάλογα με τον συνολικό αριθμό εγγραφών και τον αριθμό των νημάτων. Κατά τη διάρκεια της εκτέλεσης, δημιουργούνται τα κλειδιά και οι τιμές για τις εγγραφές που θα εισαχθούν στη βάση δεδομένων. Τα κλειδιά δημιουργούνται είτε τυχαία (αν r = 1) είτε με κανονική σειρά (αν r = 0). Οι τιμές των εγγραφών δημιουργούνται με βάση τα κλειδιά. Κάθε εγγραφή προστίθεται στη βάση δεδομένων χρησιμοποιώντας τη συνάρτηση db_add. Στη συνέχεια, αυξάνεται ο μετρητής total_writes_thread κατά ένα. Κατά τη διάρκεια της εκτέλεσης, ανά τόσες εγγραφές εκτυπώνεται ένα μήνυμα για το πόσες εγγραφές έχουν ήδη ολοκληρωθεί. Στο τέλος, το νήμα εκτυπώνει ένα μήνυμα που αναφέρει τον αριθμό των κλειδιών που έχουν εισαχθεί στη βάση δεδομένων, συνοδευόμενο από το ID του νήματος που το εκτέλεσε. Η συνάρτηση επιστρέφει NULL, καθώς δεν χρειάζεται να επιστρέψει κάποια τιμή.

- Στο db.c:
  - Το συγκεκριμένο αρχείο περιέχει τις λειτουργίες που εκτελούνται στο παρασκήνιο και είναι η κινητήριος δύναμη της μηχανής.
  - Στην αρχή του κώδικα ορίζουμε τις παρακάτω μεταβλητές και αρχικοποιούμε στατικά μια κλειδαριά και δύο συνθήκες μεταβλητής pthreads(επεξηγηματικά σχόλια υπάρχουν στην εικόνα 14).

```
int keys_get=0;//metrhths gia to plhthos twn keidiwn poy exoun vrethei
int keys_add=0;//metrhths gia to plhthos twn kleidiwn poy exoyn eisaxthei apo writers
int readcount=0;//metrhths gia ton arithmo twn readers
int writer_try_for_writing=0;//mia metavlhth pou thn theloume an yparxei prospatheia gia writing
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;//statikh arxikopoihsh mutex
pthread_cond_t readers_cond = PTHREAD_COND_INITIALIZER;//statikh arxikopoihsh metavlhths synthikhs gia tous readers
pthread_cond_t writers_cond = PTHREAD_COND_INITIALIZER;//statikh arxikopoihsh metavlhths synthikhs gia tous writers
```

-Εικόνα 14-

  - Η συνάρτηση db_add συντάχθηκε όπως φαίνεται στην εικόνα 15.

```c
int db_add(DB* self, Variant* key, Variant* value) {
    int return_value;

    pthread_mutex_lock(&mutex);// kleidoma tou mutex prin tin prosvash se krisimes perioxes
    writer_try_for_writing = 1;//orismos metavlhths gia na mas deixnei oti thelei na grapsei
    while (readcount > 0) {
        pthread_cond_wait(&writers_cond, &mutex);//edw exoume anamonh mexri na mhn yparxoun alloi readers
    }

    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);
    }

    return_value = memtable_add(self->memtable, key, value);
    printf("Thread with id %lu wrote %d key\n", pthread_self(), keys_add);
    keys_add++;//auksanoume ta kleidia pou grapsei to sygkekrimeno thread
    writer_try_for_writing= 0;//allazoume thn metavlhth se mhden efoson teleiwse to write
    pthread_cond_broadcast(&readers_cond);//ksypname olous tous readers
    pthread_cond_broadcast(&writers_cond);//ksypname olous toys writers
    pthread_mutex_unlock(&mutex);//kseklidwnoume to mutex kai apoxwroume apo krisimh perioxh
    return return_value;
}
```

-Εικόνα 15-

Σε αυτή την συνάρτηση το νήμα κλειδώνει το mutex προκειμένου να προστατεύσει την πρόσβαση σε κρίσιμες περιοχές κώδικα. Ορίζει τη μεταβλητή writer_try_for_writing σε 1, υποδεικνύοντας ότι θέλει να εκτελέσει μια εγγραφή. Ελέγχει αν υπάρχουν άλλα νήματα ανάγνωσης που βρίσκονται σε λειτουργία. Αν υπάρχουν, το νήμα περιμένει (με τη χρήση της συνθήκης writers_cond) μέχρι να ολοκληρωθούν όλες οι αναγνώσεις πριν προχωρήσει στην εγγραφή. Αν η μνήμη προς εγγραφή χρειάζεται συμπύκνωση (compaction), εκτελείται η διαδικασία συμπύκνωσης μνήμης. Αυτό επιτυγχάνεται μέσω της συνάρτησης memtable_needs_compaction. Αν υπάρχει ανάγκη, η μνήμη συμπτύσσονται μέσω των συναρτήσεων sst_merge και memtable_reset. Εκτελείται η εγγραφή στη μνήμη μέσω της συνάρτησης memtable_add. Τυπώνεται ένα μήνυμα επιβεβαίωσης για το κλειδί που εισήχθη και αυξάνεται ο μετρητής keys_add. Η μεταβλητή writer_try_for_writing ορίζεται σε 0, υποδεικνύοντας ότι το νήμα ολοκλήρωσε την εγγραφή. Με τη χρήση των συνθηκών readers_cond και writers_cond, το νήμα ειδοποιείται ότι ολοκλήρωσε την εγγραφή του και ότι μπορούν να συνεχιστούν οι αναγνώσεις και οι εγγραφές από τα υπόλοιπα νήματα. Τέλος, το mutex ξεκλειδώνεται προκειμένου να επιτραπεί η πρόσβαση άλλων νημάτων στις κρίσιμες περιοχές του κώδικα.

```
int db_get(DB* self, Variant* key, Variant* value) {

    int return_value;
    pthread_mutex_lock(&mutex);//kleidwma tou mutex gia krisimh perioxh
    while (writer_try_for_writing) {
        pthread_cond_wait(&readers_cond, &mutex);////perimenoume mexri o writer na teleiwsei
    }
    readcount++;//aukshsh twn anagnwstwn
    pthread_mutex_unlock(&mutex);//ksekleidwma tou mutex afou fygame apo krisimh perioxh
    if (memtable_get(self->memtable->list, key, value) == 1) {
        printf("Thread with id %lu found %d key in MEMTABLE\n", pthread_self(), keys_get);
        keys_get++;//aukshsh tou metrhth gia ta kleidia pou exoun vrethei apo anagnwsth
        return_value = 1;
    } else {//an den to vroume sto memtable to anazhtoume sto stt
        return_value = sst_get(self->sst, key, value);
        printf("Thread with id %lu found %d key in SST\n", pthread_self(), keys_get);
        keys_get++;////aukshsh tou metrhth gia ta kleidia pou exoun vrethei apo anagnwsth
    }
    pthread_mutex_lock(&mutex);//kleidwma tou mutex prin thn meiwsh tou readcount
    readcount--;//meiwsh twn readers
    if (readcount== 0) {
        pthread_cond_signal(&writers_cond);//an twra oi readers einai 0 tote kanoume signal wste na skypnhsoume writers
    }
    pthread_mutex_unlock(&mutex);//ksekleidwma tou mutex afou vgainoume apo krisimh perioxh
    return return_value;
}
```

-Εικόνα 16-

Καταρχάς, το νήμα κλειδώνει το mutex προκειμένου να προστατεύσει την πρόσβαση σε κρίσιμες περιοχές κώδικα. Έπειτα, ελέγχει αν υπάρχει κάποιο άλλο νήμα εγγραφής σε λειτουργία. Αν υπάρχει, το νήμα περιμένει (χρησιμοποιώντας τη συνθήκη readers_cond) μέχρι να ολοκληρωθούν όλες οι εγγραφές πριν προχωρήσει στην ανάγνωση. Αυξάνει τον μετρητή readcount για να δείξει ότι ένα νέο νήμα ανάγνωσης έχει ξεκινήσει τη λειτουργία του. Το νήμα προσπαθεί να διαβάσει το κλειδί από την μνήμη (memtable) χρησιμοποιώντας τη συνάρτηση memtable_get. Αν το κλειδί βρεθεί στη μνήμη, εκτυπώνεται ένα μήνυμα επιβεβαίωσης και ο μετρητής keys_get αυξάνεται. Αν το κλειδί δε βρεθεί στη μνήμη, το νήμα προσπαθεί να το βρει στη δευτερεύουσα δομή (SST). Το αποτέλεσμα της αναζήτησης επιστρέφεται από τη συνάρτηση sst_get. Μειώνει τον μετρητή readcount για να υποδείξει ότι ολοκλήρωσε τη λειτουργία του ως νήμα ανάγνωσης. Εάν ο μετρητής readcount γίνει μηδέν, το νήμα ειδοποιεί τα νήματα εγγραφής ότι μπορούν να συνεχίσουν τη λειτουργία τους, χρησιμοποιώντας τη συνάρτηση pthread_cond_signal. Τέλος, το mutex ξεκλειδώνεται προκειμένου να επιτραπεί η πρόσβαση άλλων νημάτων στις κρίσιμες περιοχές του κώδικα.

Επιβεβαίωση ορθής λειτουργίας:

Εκτελούμε την εντολή ./kiwi-bench rw 100 4 50 για να συγκρίνουμε με τις αρχικές διαπιστώσεις και παίρνουμε τα ακόλουθα αποτελέσματα. Ουσιαστικά, ζητάμε να έχουμε 4 threads με 50% read και 50% write.

Για 100 ζεύγη(εικόνα 17):

```
[3534] 01 Apr 16:37:00.847 . sst_loader.c:211 Bloom offset 3191 size: 124
[3534] 01 Apr 16:37:00.847 . sst.c:636 Compaction of 50 elements finished
[3534] 01 Apr 16:37:00.847 - file.c:170 Truncating file testdb/si/manifest to 44 bytes
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 0 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 1 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 2 [  1 files,   3 KiB  ]---
[3534] 01 Apr 16:37:00.847 . sst.c:55 Metadata filenum:0 smallest: key-0 largest: key-9
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 3 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 4 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 5 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
[3534] 01 Apr 16:37:00.847 . log.c:46 Removing old log file testdb/si/-1.log
[3534] 01 Apr 16:37:00.847 . sst.c:170 Merge successfully completed. Releasing the skiplist
[3534] 01 Apr 16:37:00.847 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
[3534] 01 Apr 16:37:00.847 - sst.c:176 Exiting from the merge thread as user requested
[3534] 01 Apr 16:37:00.848 - file.c:170 Truncating file testdb/si/manifest to 44 bytes
[3534] 01 Apr 16:37:00.848 . log.c:46 Removing old log file testdb/si/0.log
+----------------------------+----------------+----------------------------+------------------+
|Threaded-ReadWrite (done:100): 0.000000 (found:32548) sec/op; inf ops/sec(estimated); cost:0.000(sec);
```

-Εικόνα 17-

Για 100.000 ζεύγη(εικόνα 18):

```
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 1 [  1 files,   2 MiB  ]---
[3542] 01 Apr 16:37:54.408 . sst.c:55 Metadata filenum:11 smallest: key-0 largest: key-9999
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 2 [  1 files,   3 KiB  ]---
[3542] 01 Apr 16:37:54.408 . sst.c:55 Metadata filenum:0 smallest: key-0 largest: key-9
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 3 [  0 files,   0 bytes]---
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 4 [  0 files,   0 bytes]---
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 5 [  0 files,   0 bytes]---
[3542] 01 Apr 16:37:54.408 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
[3542] 01 Apr 16:37:54.408 . log.c:46 Removing old log file testdb/si/11.log
[3542] 01 Apr 16:37:54.408 . sst.c:170 Merge successfully completed. Releasing the skiplist
[3542] 01 Apr 16:37:54.408 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
[3542] 01 Apr 16:37:54.408 - sst.c:176 Exiting from the merge thread as user requested
[3542] 01 Apr 16:37:54.408 - file.c:170 Truncating file testdb/si/manifest to 224 bytes
[3542] 01 Apr 16:37:54.410 . log.c:46 Removing old log file testdb/si/12.log
+----------------------------+----------------+----------------------------+------------------+
|Threaded-ReadWrite (done:100000): 0.000040 (found:81253) sec/op; 400000.0 ops/sec(estimated); cost:1.000(sec);
```

-Εικόνα 18-

Για 1.000.000 ζεύγη(εικόνα 19):

```
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 1 [  1 files,   1 MiB  ]---
[3558] 01 Apr 16:38:40.359 . sst.c:55 Metadata filenum:163 smallest: key-232615 largest: key-492504
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 2 [  1 files,  31 MiB  ]---
[3558] 01 Apr 16:38:40.359 . sst.c:55 Metadata filenum:157 smallest: key-0 largest: key-99999
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 3 [  0 files,   0 bytes]---
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 4 [  0 files,   0 bytes]---
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 5 [  0 files,   0 bytes]---
[3558] 01 Apr 16:38:40.359 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
[3558] 01 Apr 16:38:40.359 . log.c:46 Removing old log file testdb/si/120.log
[3558] 01 Apr 16:38:40.359 . sst.c:170 Merge successfully completed. Releasing the skiplist
[3558] 01 Apr 16:38:40.359 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
[3558] 01 Apr 16:38:40.359 - sst.c:176 Exiting from the merge thread as user requested
[3558] 01 Apr 16:38:40.360 - file.c:170 Truncating file testdb/si/manifest to 232 bytes
[3558] 01 Apr 16:38:40.370 . log.c:46 Removing old log file testdb/si/121.log
+----------------------------+----------------+----------------------------+------------------+
|Threaded-ReadWrite (done:1000000): 0.000048 (found:532711) sec/op; 333333.3 ops/sec(estimated); cost:12.000(sec);
```

-Εικόνα 19-

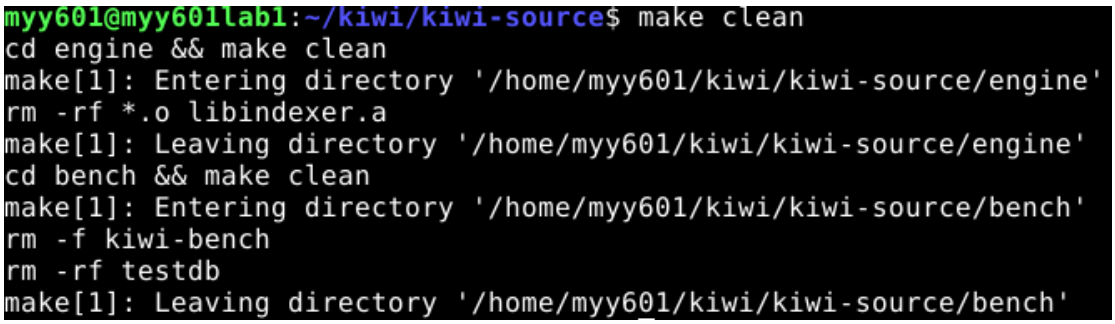
Σημείωση: Δυστυχώς υπάρχει ένα πρόβλημα με την μεταβλητή found και εμφανίζονται απρόσμενα αποτελέσματα και λόγω περιορισμένου χρόνου δεν διευθετήθηκε.

Τελικά αποτελέσματα

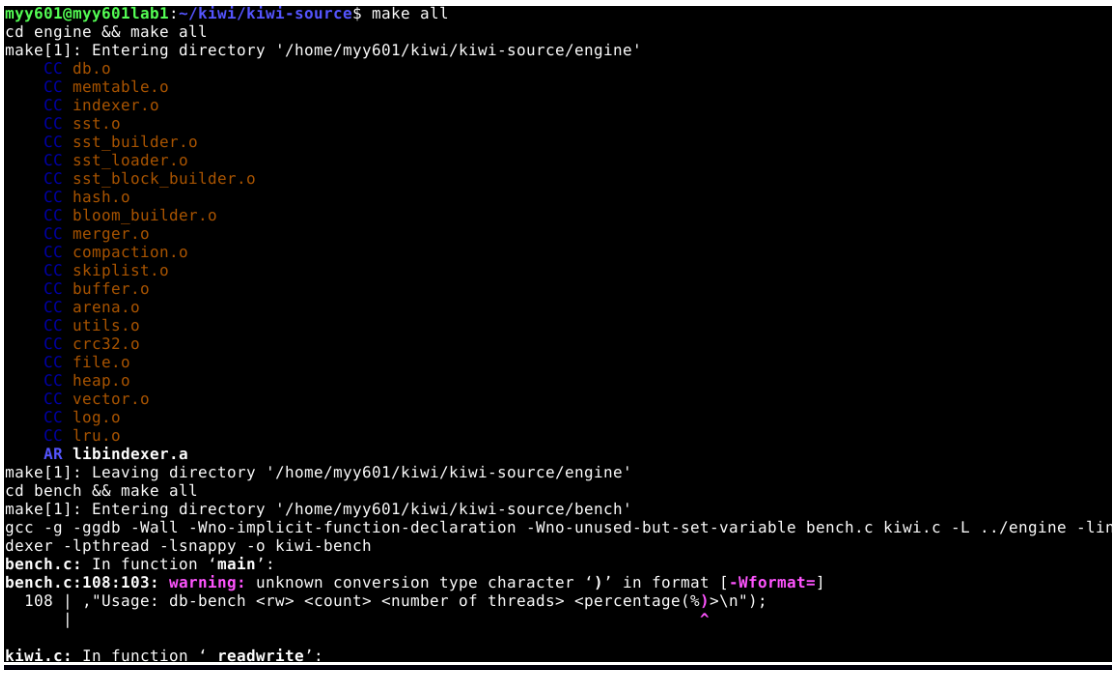
Όπως μπορούμε να παρατηρήσουμε με πολλαπλά παραδείγματα στο κώδικα, η υλοποίηση αναγνωστών-γραφέων λειτουργεί σωστά. Με τη πρώτη ματιά, οι λειτουργίες φαίνεται να τρέχουν ταχύτερα. Αυτό όμως θα διαπιστωθεί παρακάτω στα συμπεράσματα.

Εντολές make

Παρακάτω παρατίθενται τα αποτελέσματα της εκτέλεσης της εντολής make clean και make all στις εικόνες 20 και 21 από το τερματικό.



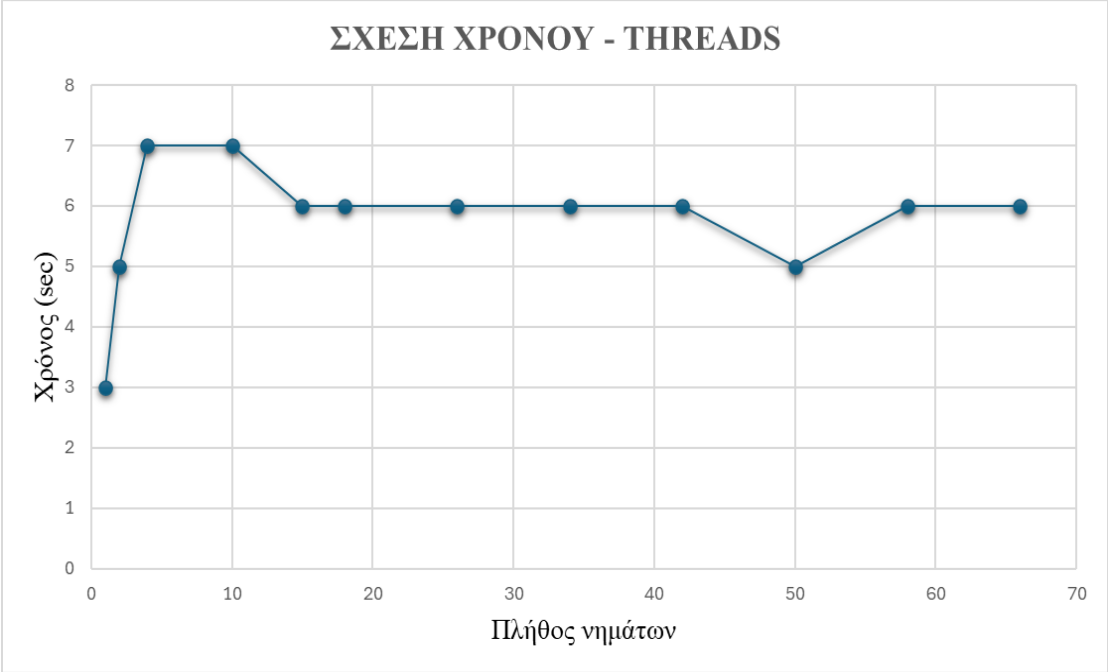
-Εικόνα 20-



-Εικόνα 21-

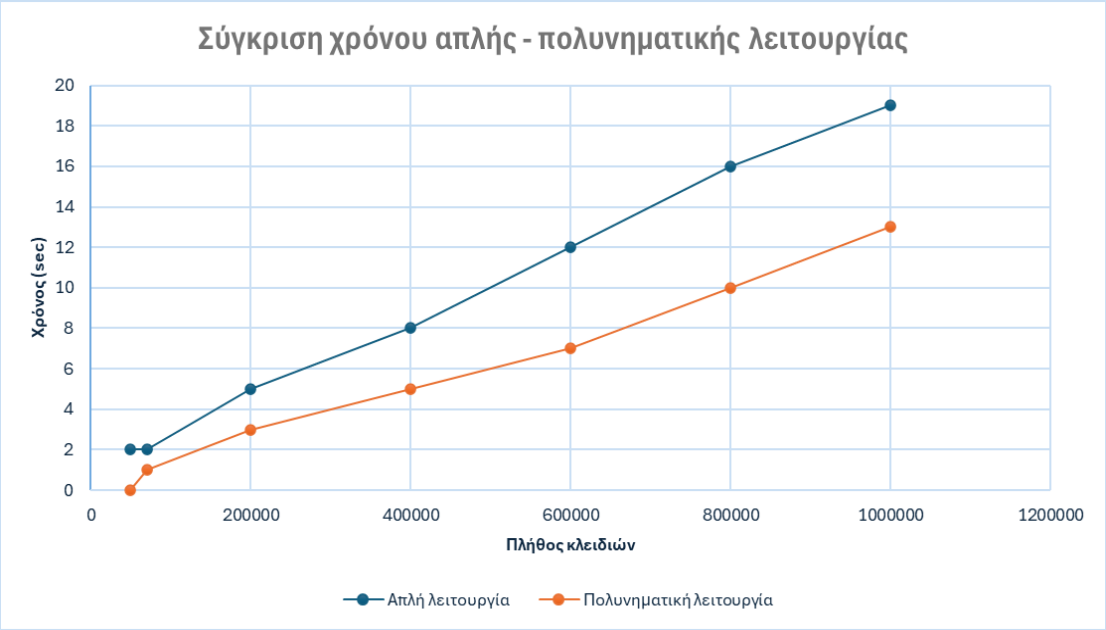
Συμπεράσματα

Χρόνοι - νήματα



Στο παραπάνω διάγραμμα βλέπουμε πως εξελίσσεται ο χρόνος σε σχέση με τα νήματα έχοντας ποσοστό read 50% και write 50% και γράφοντας 500.000 κλειδιά. Παρατηρούμε πως για πολύ λίγα νήματα (2) ο χρόνος βρίσκεται στα 3 seconds. Για 10 νήματα βλέπουμε ότι η τιμή πάει στο peak 7 seconds, όμως συμβαίνει κάτι περίεργο για 50 νήματα. Ο χρόνος πέφτει πάλι στα 6 seconds. Αυτό μπορεί να συμβαίνει διότι ο επεξεργαστής εκείνη τη στιγμή μπορεί να διαχειρίστηκε πιο αποδοτικά τα mutexes. Πολλοί θα περιμένανε με την αύξηση των νημάτων θα μειωνόταν και ο χρόνος εκτέλεσης, αυτό όμως θα ήταν λάθος να το πούμε με σιγουριά, διότι δεν γνωρίζουμε πως θα αντιδράσει ο επεξεργαστής και το λειτουργικό σύστημα σε κάθε περίπτωση.

Διαφορές απλής – πολυνηματικής λειτουργίας



Στο παραπάνω διάγραμμα φαίνονται οι καμπύλες της απλής λειτουργίας (μπλε γραμμή), αλλά και της πολυνηματικής λειτουργίας (πορτοκαλί γραμμή) σύμφωνα με το πλήθος των κλειδιών. Για την πολυνηματική λειτουργία περνάμε ως παράμετρο 4 threads, 50% read και 50% write. Για την απλή λειτουργία, αρχικά κάνουμε write το πλήθος των κλειδιών που επιθυμούμε και έπειτα read ώστε να υπολογίσουμε το χρόνο που χρειάζεται για τη λειτουργία εγγραφής και ανάγνωσης. Παρατηρούμε ότι με την πολυνηματική λειτουργία έχουμε ταχύτερους χρόνους σε σχέση με την απλή. Πιθανότατα ο μερικός παραλληλισμός των εντολών να μας επιφέρει ταχύτερες επιδόσεις.