Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Ακαδημαϊκό Έτος 2022-2023

ΜΥΥ601 Λειτουργικά Συστήματα

1η Εργαστηριακή Άσκηση

Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

ΟΜΑΔΑ

Αρχοντής Νέστωρας - 4747
Σπυρίδων Χαλιδιάς - 4830

Περιεχόμενα

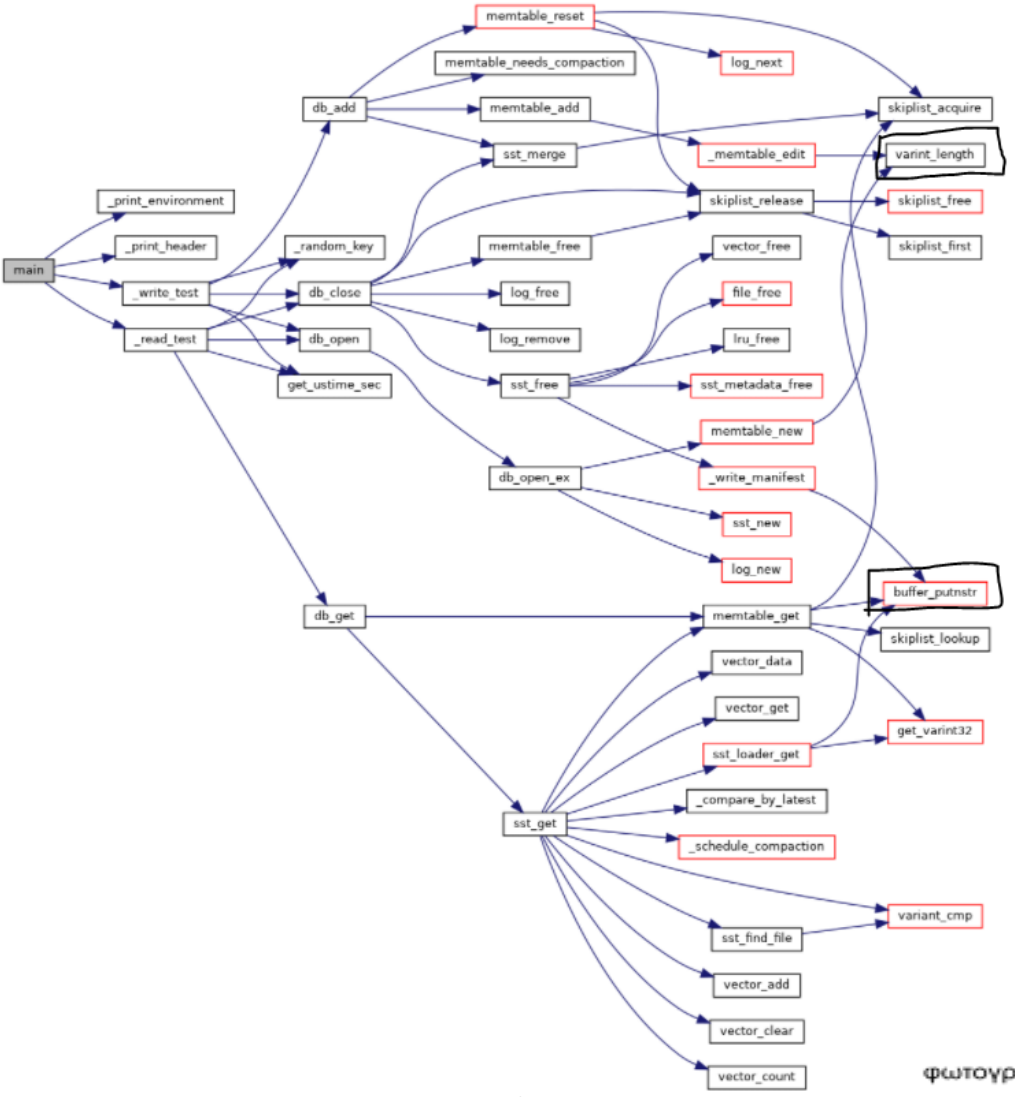
Τρόπος Σκέψεις

Εισαγωγή

Στόχος αυτής της εργασίας είναι η υλοποίηση πολυνηματικής λειτουργίας στη μηχανή αποθήκευσης Kiwi. Για να συμβεί αυτο ακολουθήσαμε μία σειρά από βήματα που παρουσιάζονται στη συνέχεια.

Εύρεση κρίσιμων περιοχών

Αρχικά πρέπει να βρεθούν οι κρίσιμες περιοχές δηλαδή οι περιοχες που θα έχουν πρόβλημα όταν δέχονται πολλά νήματα.Παρατηρώντας το διάγραμμα της main βλεπουμε περιοχές που ενδέχεται να υπάρχει πρόβλημα καθώς τις διατρέχουν τα νήματα της write της read. Κάποιες από αυτές τις περιοχές είναι οι buffer_putnstr, varint_length (φωτογραφία 1). Έπειτα ελέγχουμε αυτή τη σκέψη δημιουργώντας νήματα για τη write και τη read. Το πρόγραμμα αδυνατεί να τρέξει και πετάει error. Μέσα από το debugger (gdb) βλέπουμε ότι το σημείο που έχει πρόβλημα είναι το buffer_extend_by (φωτογραφία 2) το οποίο ανήκει ανήκει σε αυτά που προβλέψαμε. Πρέπει λοιπόν η υλοποίηση να λύνει τέτοια προβλήματα.



φωτογραφία 1

φωτογραφία 2

Η ιδέα για την υλοποίηση του πολυνηματισμού

Η πρώτη ιδέα είναι να βρούμε όλες τις κρίσιμες περιοχές και να φτιάξουμε εκεί τα πολλαπλά νήματα με mutexes και condition variables. Αυτή η ιδέα φαίνεται αρκετά καλή. Όμως αυτή η υλοποίηση έχει μεγάλη δυσκολία. Αποφασίσαμε έτσι να συνεχίσουμε δουλεύοντας πιό γενικά. Η υλοποίηση που κάναμε έγινε μέσα στον φάκελο Engine και συγκεκριμένα στο αρχείο db.c. Η μέθοδος που χρησιμοποιήσαμε είναι η condition variables. Η ιδέα είναι ότι θα γίνουν create και τα δύο νηματα write και create. Έπειτα το νήμα της read θα κλειδώσει και θα περιμένει την write για να συνεχίσει. Όταν η write τελειώσει τότε η write θα συνεχίσει και θα κάνει τον ταυτοχρονισμό με τα υπόλοιπα write.

Τροποποιήσεις στον πηγαίο κώδικα

Εισαγωγή

Στον πηγαίο κώδικα έχουν γίνει αλλαγές σε τρία σημεία, στα αρχεία bench.c, kiwi.c και db.c.

Περιληπτικά, προσθέσαμε την δυνατότητα να γίνονται παράλληλα οι εγγραφές και τα διαβάσματα αιτήσεων, με την τεχνική του πολυνηματισμού και του αμοιβαίου αποκλεισμού. Αυτό επιτεύχθηκε με την δημιουργία της λειτουργίας writeread ώπου ο χρήστης μπορεί να την καλέσει από το τερματικό. Επίσης, μετά από ορισμένες επεκτάσεις, στο τερματικό πλέον μπορούμε να βάλουμε, εκτός από τον αριθμό των αιτήσεων, και τον αριθμό των νημάτων καθώς και τα ποσοστά για την κατανομή των αιτήσεων στις εγγραφές ή στις αναγνώσεις και τέλος μπορούμε να βάλουμε την τυχαιότητα των τιμών των κλειδιών. Θα δούμε παρακάτω πιο αναλυτικά, στην ενότητα με τα Παραδείγματα Εκτέλεσης στο Τερματικό, τι χρειάζεται η κάθε περίπτωση κλήσης μιας λειτουργίας.

Δεν πρέπει να παραλείψουμε ότι όλες οι προηγούμενες λειτουργίες του ξεχωριστού write και read, λειτουργούν ορθά όπως λειτουργούσαν στον πρωταρχικό κώδικα που μας είχε δοθεί, απλά τώρα καλούμε ένα νήματος ανά περίπτωση. Έπειτα από προσωπικούς ελέγχους που εκτελέσαμε προκειμένου να βεβαιωθούμε για την ορθότητα της σκέψης μας, καταλήξαμε στο συμπέρασμα ότι πρακτικά δεν αλλάζει κάτι γιατί και παλιά υποσεινήδητα γινόταν η ίδια διαδικασία.

Ας εξηγήσουμε τώρα παρακάτω, ορισμένα κομμάτια από τον κώδικα που τροποποιήσαμε-προσθέσαμε, με σκοπό την υλοποίηση όλων των παραπάνω. Σχεδόν κάθε γραμμή κώδικα που προσθέσαμε, συνοδεύεται από επεξηγηματικά σχόλια για την κατανόησή του.

bench.c

Με λίγα λόγια, οι βασικές αλλαγές που έγινα στο bench.c αρχείο, αφορούν την δημιουργία νημάτων και την εμπλούτιση της main μας με παραπάνω παραμέτρους.

Ορίζουμε τις μεταβλητές που θα χρησιμοποιήσουμε παρακάτω και φτιάχνουμε μία δομή που θα μας βοηθήσει στην δημιουργία των νημάτων ως ορίσματα στις συναρτήσεις που θα καλούν.

```
6  pthread_t tid_write; //kanoume global metavlhth to id tou nhmatos write giati to theloume sto arxeio db.c
7
8  long int count; //krataei ton ariumo tvn write kai read pou thelei o xrhsths
9
10 void _write_test(void *arg); //yparxei sto kiwi.c kai einai protypo
11
12 void _read_test(void *arg); //yparxei sto kiwi.c kai einai protypo
13
14 extern int wr; //to pernoume san global apo to db.c gia na boresoume na epitrecoume sto thread read na trexei kai na min perimenei
15
16 long int *arguments; //enas diktis pou krataei to orismata apo tin main gia na min xathoun argotera me to aplo argv
17
18 //gia na dexetai pollapla orismata sto create afou exoume ftiajei nhmata
19 struct data {
20   long int countt;
21   int rr;
22 };
```

Ορίζουμε και αρχικοποιούμε ορισμένες μεταβλητές στην main, ενώ επίσης ορίζουμε μία δομή για το write και μία δομή για το read.

```
int main(int argc,char** argv)
{
  struct data thread_args_write; //ginetai arxikopoihsh domvn gia na xreisimopoihthoun san orismata sthn dhmioyrgia tou nhmatos write
  struct data thread_args_read; //ginetai arxikopoihsh domvn gia na xreisimopoihthoun san orismata sthn dhmioyrgia tou nhmatos read

  pthread_t tid[100]; //arxikopoihsh enos pinaka pou krataei ta id tvn nhmatvn(protinete mexri 100 nhmata giati perissotera den exei noima, alla ginetai na boun kai parapanv nhmata)

  int r = 0; //to r einai ypeyuyno gia to an ua kalesei thn _random_key

  count = 0; //mia metavliti opou melodika tha krataei ton arithmo tvn aithsevn pou tha thetei o xristis

  int num_of_treads; //mia metavliti opou melodika tha krataei ton arithmo tvn nimatvn pou tha thetei o xristis

  long int percentage_write = 0; //mia metavliti opou melodika tha krataei to pososto tvn aithsevn pou afora ta writes pou tha thetei o xristis
  long int percentage_read = 0; //mia metavliti opou melodika tha krataei to pososto tvn aithsevn pou afora ta reads pou tha thetei o xristis

  arguments = (long int*) malloc((4)*sizeof(long int)); //desmeuume xoro gia enan dynamiko pinaka gia tin apothikeusi ton orismaton tis main
```

Γενικά, έχουν γίνει αρκετοί έλεγχοι κατα μήκος όλου του κώδικα, προκειμένου να εντοπίσουμε κάποιες λανθασμένες κλίσης στο τερματικό και να προτείνουμε μία πιο σωστή σύνταξη. Επίσης ανάλογα με την λέξη του τελευταίου ορίσματος του τερματικού, θέτουμε την σωστή τιμή που πρέπει να έχει η μεταβλητή r, η οποία είναι υπεύθυνη για την τυχαιότητα ή μη των τιμών των κλειδιών.

```
  srand(time(NULL));
  if (argc <= 3) {
    fprintf(stderr,"Usage: db-bench <write | read> <count> <random> OR\n <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKeys> OR\n
    exit(1);
  }

  if (strcmp(argv[argc-1],"RandomKeys") == 0) { //an sto telos tvn orismatvn ths main valoume RandomKeys tha trejei gia tyxaia kleidia
    r = 1;
  }
  else if(strcmp(argv[argc-1],"noRandomKeys") == 0) { //allivs an valoume noRandomKeys tha trexei me thn seira ta kleidia 0,1,2,...
    r = 0;
  }
  else {
    fprintf(stderr,"Usage: db-bench <write | read> <count> <random> OR\n <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKeys> OR\n
      exit(1);
  }
```

Εδώ υλοποιούμε την write και την read κλίση, δύο κλήσεις που δεν μας απασχολεί η πολυνηματική τους λειτουργία, αφού χρειάζονται ένα νήμα σε κάθε κλίση, άρα πρακτικά μπορεί να χρησιμοποιήσει την αρχική συνάρτηση και θα είναι σαν να χρησιμοποιεί ένα νήμα.

```
  if (strcmp(argv[1], "write") == 0) {
    wr = 0; //gia na apofigoume ton tautoxronismo
    if (argc > 4){
      fprintf(stderr,"Usage: db-bench <write | read > <count>\n");
      exit(1);
    }
    count = atoi(argv[2]); //to 3o orisma einai o arithmos ton etiseon
    _print_header(count);
    _print_environment();
    _write_test_1(count, r); //kalei tin apli arxiki synartisi(praktika trexei se 1 nhma)

    //vazoume stin 1h thesh tou pinaka me ta id tvn threads to torino thread(einai perissotero gia na iparxei mia oraia domi sta apotelesmata)
    tid[0] = pthread_self();
  } else if (strcmp(argv[1], "read") == 0) {
    wr = 0; //gia na apofigoume ton tautoxronismo
    if (argc > 4){
      fprintf(stderr,"Usage: db-bench <write | read > <count>\n");
      exit(1);
    }
    count = atoi(argv[2]); //to 3o orisma einai o arithmos ton etiseon
    _print_header(count);
    _print_environment();
    _read_test_1(count, r); //kalei tin apli arxiki synartisi(praktika trexei se 1 nhma)

    //vazoume stin 1h thesh tou pinaka me ta id tvn threads to torino thread(einai perissotero gia na iparxei mia oraia domi sta apotelesmata)
    tid[0] = pthread_self();
```

Εδώ υλοποιούμε την πολυνηματική λειτουργία του προγράμματός μας, που είναι ο βασικός σκοπός της άσκησης. Δημιουργούμε μία καινούργια κλίση, την writeread, που θα μπορεί να καλεί ο χρήστης από το τερματικό. Δύο είναι οι βασικοί τρόποι για να κλιθεί, με ή χωρίς ποσοστά, αφού πρώτα έχουμε ορίσει πόσα νήματα read θα θέλαμε. Η ύπαρξη των ποσοστών είναι υπεύθυνη για την καταμέριση των αιτήσεων ανάμεσα στο νήμα write και read. Επίσης εδώ γεμίζουμε και τον πίνακα arguments, έναν πίνακα σημαντικό, για να περάσουμε στο αρχείο kiwi.c, τα δεδομένα που θα έχουμε λάβει από το τερματικό, αφού θα έχουν υποστεί αρχικά την απαραίτητη επεξεργασία που βλέπουμε στον παρακάτω κώδικα.

```
153   } else if (strcmp(argv[1], "writeread") == 0) {
154     if (atoi(argv[3]) == 0 || (argc != 5 && argc != 7)){
155       fprintf(stderr,"Usage: db-bench <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKeys> OR\n <writeread> <count>
156       exit(1);
157     }else if (argc == 5){
158       percentage_write = 100;
159       percentage_read = 100;
160     }else if (argc == 7){
161       if (atoi(argv[4]) != 0 || atoi(argv[5]) != 0){
162         percentage_write = atoi(argv[4]);
163         percentage_read = atoi(argv[5]);
164       } else {
165         fprintf(stderr,"Usage: db-bench <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKeys> OR\n <writeread> <count>
166         exit(1);
167       }
168     }else{
169       fprintf(stderr,"Usage: db-bench <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKeys> OR\n <writeread> <count>
170       exit(1);
171     }
172
173     num_of_treads = atoi(argv[3]); //to 4o orisma einai o arithmos ton nimaton
174     //gia na apofigoume ton tautoxronismo se autin tin periptosi
175     if(num_of_treads==1){
176       wr=0;
177     }
178
179     thread_args_write.rr = r; //dinoume timh sto ena apo ta dyo orismata ths _write_test
180     thread_args_read.rr = r; //dinoume timh sto ena apo ta dyo orismata ths _read_test
181
182     //mia morfi gia to ti exei mesa tou to arguments = [write.r, write.count, read.r, read.count]
183     arguments[0] = thread_args_write.rr;
184     arguments[2] = thread_args_read.rr;
185
186     count = atoi(argv[2]); //to 3o orisma einai o arithmos ton etiseon
187
188     thread_args_write.countt = count*(percentage_write/(float)100); //dinoume timh sto allo orisma ths _write_test
189     thread_args_read.countt = (count*(percentage_read/(float)100))/num_of_treads; //dinoume timh sto allo orisma ths _read_test
190     arguments[1] = thread_args_write.countt;
191     arguments[3] = thread_args_read.countt;
192
193     _print_header(count);
194     _print_environment();
```

Αρχικά δημιουργούμε ένα νήμα για το write, εφόσον τα νήματα που θέλουμε είναι περισσότερα από 1. Στην άλλη περίπτωση, δηλαδή την περίπτωση που θα θέλουμε ένα νήμα, θα καλέσουμε την απλή αρχική μας συνάρτηση όπου θα είναι πρακτικά σαν να καταλαμβάνει ένα νήμα. Αυτή την διαδικασία αναγκαστήκαμε να την κάνουμε γιατί ο ταυτοχρονισμός που έχουμε υλοποιήσει λειτουργεί βέλτιστα για όταν μας ζητείται πάνω από 2 νήματα. Είναι λογικό ότι με ένα νήμα δεν μπορεί και δεν χρειάζεται να κάνει ταυτοχρονισμό, ειδικά στην περίπτωση που το τρέχουμε σε μηχανήματα με έναν πυρήνα. Συνεχίζοντας στον κώδικα, βλέπουμε ότι δημιουργούμε όσα νήματα read μας έχει ζητήσει ο χρήστης από το τερματικό.

Έπειτα, εκτυπώνουμε έναν πολύ εύχρηστο πίνακα με τα id των νημάτων που έχουμε. Τέλος, αποδεσμεύουμε τους πόρους των νημάτων που δημιουργήσαμε και αποδεσμεύουμε και τον χώρο που είχαμε για τον δυναμικό πίνακα των arguments.

```
    // WRITE
    if (num_of_treads == 1){
      //vazoume stin 2h thesh tou pinaka me ta id tvn threads to torino thread(einai perissotero gia na iparxei mia oraia domi sta apotelesmata)
      tid[0] = pthread_self();

      //an theloume ena nhma gia to read tha kalesei aytin tin sinartisi poy praktika einai les kai trexei se ena nhma
      _write_test_1(arguments[1], arguments[0]);
    }else{
      //dhmiourgoume to nhma tou write
      pthread_create(&tid[0], NULL, _write_test, (void *) &thread_args_write);
      tid_write = tid[0]; //apothhkeboume to id tou nhmatos write sthn global metavlhth tid_write
    }

    // READ
    for (int i = 0; i < num_of_treads; i++){
      pthread_create(&tid[i+1], NULL, _read_test, (void *) &thread_args_read); //dhmiourgoume to nhma tou read
    }
  }
   else {
    fprintf(stderr,"Usage: db-bench <write | read> <count> <random> OR\n <writeread> <count> <numOfReadThreads> <noRandomKeys | RandomKey> OR\n
    exit(1);
  }

  //kanoume print ton pinaka me ta id tvn threads pou exoume
  printf("__ArrayOfThreadsID: \n");
  printf("___WriteThread-1__%d\n", tid[0]);
  for (int i=1; i<num_of_treads+1 ; i++){
    printf("___ReadThread-%d___%d\n", i, tid[i]);
  }

  for(int i = 0; i<num_of_treads+1; i++){
    pthread_join(tid[i], NULL); //anasteletai to main thread mexri na teleivsh to nhma kai apodesmeuei tous porous
  }

  free (arguments); //afou xreisimopoihsame malloc eleutheronoume ton xoro pou desmeusame gia ta arguments

  return 1;
}
```

Θα φαινόταν αρκετά χρήσιμο σε αυτό το σημείο να οπτικοποιήσουμε τι περιέχει ο πίνακας arguments, για την μεγαλύτερη κατανόηση του κώδικα. Δημιουργήθηκε αυτός ο πίνακας, προκειμένου να εξασφαλίσουμε την βαθιά αποθήκευση των ορισμάτων από το τερματικό, σε ένα σταθερό και αμετάβλητο χώρο στην μνήμη, αφού προηγουμένως είχε παρατηρηθεί ότι η απλή χρήση του argv δεν κράταγε τις εισόδους μας με τον καλύτερο δυνατό τρόπο σε μεγάλες κλίσεις.

arguments	0	1	2	3
	flag(τυχαία ή μη κλειδιά) για τα writes	count(αρ.αιτήσεων) για τα writes	flag(τυχαία ή μη κλειδιά) για τα reads	count(αρ.αιτήσεων) για τα reads

Kiwi.c

Με λίγα λόγια, οι βασικές αλλαγές που έγιναν στο kiwi.c αρχείο, αφορούν κυρίως την _write_test και την _read_test, συναρτήσεις που θα κληθούν από την δημιουργία των νημάτων. Για να επιτευχθεί αυτό αλλάξαμε τα ορίσματα τους και δέχονται πλέον ως ορίσματα struct.

Φέρνουμε τον global πίνακα που είχαμε φτιάξει στο προηγούμενο αρχείο, το bench.c, για να το χρησιμοποιήσουμε στις δύο βασικές μας συναρτήσεις στην δημιουργία των νημάτων που θα δούμε παρακάτω. Ο ορισμός του struct δεν μας χρειάζεται τελικά στον κώδικα που ακολουθεί.

```
8 extern long int *arguments; //pairnoume apo to bench.c ton pinaka pou ftiajame oste na exoume ta sosta inputs apo ta orismata tis main
9
10 //ftiajame to struct gia na trejei to creat kai antikatasthsame to arxiko orisma count me d->countt kai to arxiko orisma r me d->rr
11 struct data {
12   long int countt;
13   int rr;
14 };
```

Αλλάξαμε τα ορίσματα, προκειμένου να χρησιμοποιήσουμε αυτές τις συναρτήσεις στην δημιουργία των νημάτων write και read αντίστοιχα.

```
16 //allajame ta orismata gia na boresoume na thn xreishmopoihsoume sto create tou antistoixou nhmatos
17 void _write_test(void *arg)
18 {
19   struct data *d = (struct data *) arg;
20
21   int i;
```

```
73 //allajame ta orismata gia na boresoume na thn xreishmopoihsoume sto create tou antistoixou nhmatos
74 void _read_test(void *arg)
75 {
76   struct data *d = (struct data *) arg;
77
78   int i;
```

Το for χρησιμοποιείτε ώστε να εκτελέσει όσες αιτήσεις του έχει ζητήσει ο χρήστης στην κάθε περίπτωση, είτε αρχικα για το write και έπειτα για το read. Επίσης, το if χρησιμοποιείτε ώστε να εφαρμόσει αν ο χρήστης του ζήτησε τυχαίες τιμές στα κλειδιά ή όχι.

```
37   start = get_ustime_sec();
38   for (i = 0; i < arguments[1]; i++) {
39     if (arguments[0]){
40       _random_key(key, KSIZE);
```

```
90   for (i = 0; i < arguments[3]; i++) {
91     memset(key, 0, KSIZE + 1);
92
93     /* if you want to test random write, use the following */
94     if (arguments[2])
95       _random_key(key, KSIZE);
```

Κάνει κάποια χρήσιμα print για τους χρόνους εκτέλεσης της κάθε διαδικασίας. Είναι και μία επιβεβαίωση ότι έχει ολοκληρωθεί η κλίση του write και η κλίση του read αντίστοιχα αν προλάβουν και εμφανιστούν αυτές οι εκτυπώσεις.

```
66   printf(LINE);
67   printf("|Random-Write (done:%ld): %.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec);\n"
68     ,arguments[1], (double)(cost / arguments[1])
69     ,(double)(arguments[1] / cost)
70     ,cost);
```

```
124  printf("|Random-Read  (done:%ld, found:%d): %.6f sec/op; %.1f reads /sec(estimated); cost:%.3f(sec)\n",
125    arguments[3], found,
126    (double)(cost / arguments[3]),
127    (double)(arguments[3] / cost),
128    cost);
129 }
```

Εδώ έχουμε αφήσει τις ίδιες, αρχικές συναρτήσεις, που τις χρειαστήκαμε στο αρχείο bench.c.

```
131 //apo edo kai kato einai o arxikos kodikas pou mas eixe dothei
132 void _write_test_1(long int count, int r)
133 {
134   int i;
135   double cost;
```

```
186 void _read_test_1(long int count, int r)
187 {
188   int i;
189   int ret;
190   int found = 0;
```

db.c

Με λίγα λόγια, οι βασικές αλλαγές που έγιναν στο db.c αρχείο, αφορούν την υλοποίηση του πολυνηματισμού. Η μέθοδος που ακολουθούμε είναι η condition variables. Θεωρούμε δύο καταστάσεις την wr=0 και την wr=1. Όταν το wr=0 γίνονται τα write ενώ όταν wr=1 γίνονται τα read. Στην αρχή το wr είναι αρχικοποιημένο στο 1. Έτσι όταν το νήμα του read φτάσει στο σημείο να κάνει open, περιμένει να τελειώσει το write που θα δώσει το σήμα στο read να συνεχίσει(εδώ εξετάζουμε την τετριμμένη περίπτωση όπου αυτή η λειτουργεία γίνεται σε έναν πυρίνα). Με αυτό τον τρόπο φτιάχνουμε ένα πρώτο κομμάτι του πολυνηματισμού. Μία άλλη περίπτωση είναι να τρέχουν παράλληλα τα read. Αυτό επιτυγχάνεται με τα mutex, έτσι ώστε να υπάρχει η κατάλληλη προτεραιότητα ανάμεσα στα read. Στο σημείο που αλλάζει η συνθήκη της condition variables δίνεται το σήμα για να “ξυπνήσουν” το επόμενο νήμα που θα περιμένει. Αυτή η διαδικασία συνεχίζεται μέχρι να τελειώσουν όλες οι διεργασίες όλων των νημάτων. Με τα σχόλια και τις εκτυπώσεις που έχουμε συμπεριλάβει στον κώδικά μας, διαφαίνεται καλύτερα η πορεία του εκάστοτε νήματος.

Γίνονται οι απαραίτητες αρχικοποιήσεις τόσο σε μεταβλητές που λειτουργούν σαν flags όσο και σε σημαντικά αντικείμενα τύπου pthread, απαραίτητα για να εργαστούμε με τα νήματα και να πετύχουμε την τεχνική πολυνηματισμού, condition variables.

```
9  ////////////////////////////////////////////////////////////////
10 extern pthread_t tid_write; //erxetai apo to bench.c kai periexei to id toy nhmatos ths write
11
12 int wr = 1; //arxikopoieitai sto 1 epeidi stin db_open_ex benei prvta to nhma read
13
14 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; //arxikopoihsh mutex
15 pthread_cond_t condition = PTHREAD_COND_INITIALIZER; //arxikopoihsh condition
16 ////////////////////////////////////////////////////////////////
```

Περικλείουμε τον κώδικα της μεθόδου db_open_ex με ειδικές συναρτήσεις που αφορούν τα νήματα, προκειμένου να κλειδώσουμε τον συγκεκριμένο κώδικα για να μην καλεστεί από κάποιο άλλο νήμα. Κάνουμε παράλληλα και κάποιους απαραίτητους ελέγχους για να διασφαλίσουμε την σωστή προτεραιότητα της κλήσης των νημάτων. Αυτό επιτυγχάνεται με το να βάζουμε, όποιο νήμα χρειάζεται, σε μία κατάσταση wait.

```
18 DB* db_open_ex(const char* basedir, uint64_t cache_size)
19 {
20 ////////////////////////////////////////////////////////////////
21
22     pthread_mutex_lock(&mutex); //kleidvnetai o parakatv kvdikas
23     if(pthread_equal(pthread_self(), tid_write)){ //elexei an to nhma pou kaleitai ekeinh thn stigmh einai to write
24       wr = 0; //an to parapanv isxyei to wr ginetai 0 pou shmainei oti trexei to nhma write
25     }
26
27 ////////////////////////////////////////////////////////////////
28     DB* self = calloc(1, sizeof(DB));
29 ////////////////////////////////////////////////////////////////
30
31     while(wr == 1){ //oso vlepei to wr na einai 1, dhladh na trexei nhma read
32       //an tipvthei auto epivevaivnei oti stin arxh exei klhthei to nhma read
33       printf("........I AM WAITIMG(CurrentThreadId: %d)........\n", pthread_self());
34       pthread_cond_wait(&condition,&mutex); //kanei wait
35     }
36     printf("........I AM RUNNING(CurrentThreadId: %d)........\n", pthread_self());
37
38 ////////////////////////////////////////////////////////////////
39     if (!self)
40         PANIC("NULL allocation");
41
42     strncpy(self->basedir, basedir, MAX_FILENAME);
43     self->sst = sst_new(basedir, cache_size);
44
45     Log* log = log_new(self->sst->basedir);
46     self->memtable = memtable_new(log);
47 ////////////////////////////////////////////////////////////////
48
49     pthread_mutex_unlock(&mutex); //xekleidvnetai o parapanv kvdikas
50     printf("........I AM HERE(CurrentThreadId: %d)........\n", pthread_self());
51
52 ////////////////////////////////////////////////////////////////
53     return self;
54 }
```

Εργαζόμαστε και εδώ με παρόμοιο τρόπο, όπως παραπάνω, περικλείοντας τον κώδικα της μεθόδου db_close με ειδικές συναρτήσεις που αφορούν τα νήματα, προκειμένου να κλειδώσουμε τον συγκεκριμένο κώδικα για να μην καλεστεί από κάποιο άλλο νήμα. Όταν κρίνεται σκόπιμο, βγάζουμε από την κατάσταση wait το προαναφερόμενο νήμα, ώστε να αρχίσει και αυτό με την σειρά του την εκτέλεση. Αυτή εδώ η μέθοδος, είναι ένα καλό σημείο να υλοποιήσουμε αυτήν την διαδικασία, αφού είναι από τις τελευταίες συναρτήσεις που καλείτε από την μέθοδο των νημάτων.

```
61 void db_close(DB *self)
62 {
63 ////////////////////////////////////////////////////////////////
64
65     pthread_mutex_lock(&mutex); //kleidvnetai o parakatv kvdikas
66     printf("____CurrentThreadId: %d\n", pthread_self());
67     wr = 2; //otan teleivsei to prvto write ta nhmata read prepei na bainoun synexeia sthn open gi auto to wr paramenei 0
68     if (wr == 2){
69       pthread_cond_signal(&condition); //to nhma write jypnaei to nhma read
70       printf("........THREAD READ IS AWAKE AND WAITTING[if the thread exists]........\n");
71     }
72     printf("____CurrentThreadId: %d\n", pthread_self());
73
74 ////////////////////////////////////////////////////////////////
75     INFO("Closing database %d", self->memtable->add_count);
76
77     if (self->memtable->list->count > 0)
78     {
79         sst_merge(self->sst, self->memtable);
80         skiplist_release(self->memtable->list);
81         self->memtable->list = NULL;
82     }
83
84     sst_free(self->sst);
85     log_remove(self->memtable->log, self->memtable->lsn);
86     log_free(self->memtable->log);
87     memtable_free(self->memtable);
88     free(self);
89 ////////////////////////////////////////////////////////////////
90
91     pthread_mutex_unlock(&mutex); //xekleidvnetai o parapanv kvdikas
92     printf("........I FINISH(CurrentThreadId: %d).........\n", pthread_self());
93
94 ////////////////////////////////////////////////////////////////
95 }
```

Γραφήματα & Στατιστικά

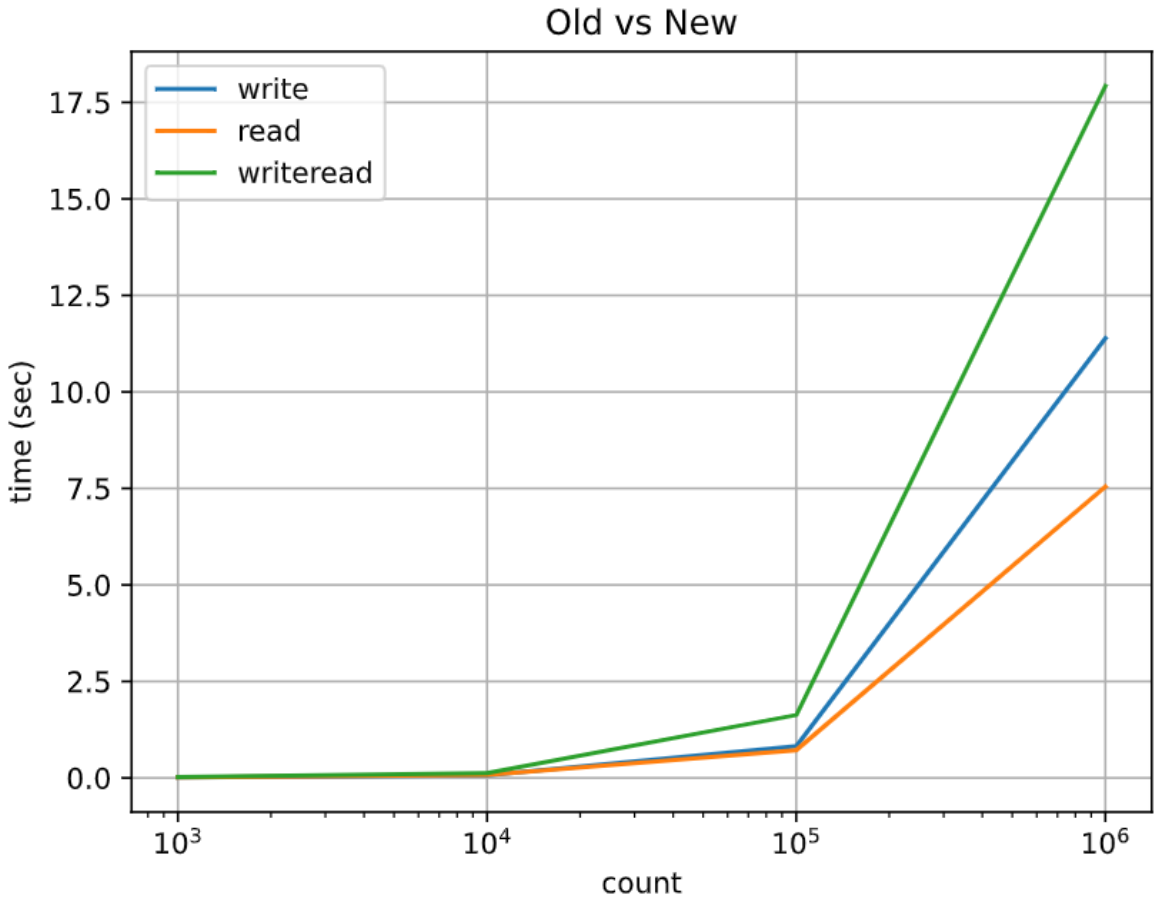
Εισαγωγή

Πριν ξεκινήσουμε την παρουσίαση των γραφημάτων να αναφέρουμε ότι οι μετρήσεις για κάθε γράφημα και για κάθε τιμή (εκτός από μία συνάρτηση) είναι ο μέσος όρος 5 διαφορετικών μετρήσεων. Αυτό υλοποιήθηκε με την βοήθεια ορισμένων script που συντάξαμε. Ο σχεδιασμός των γραφικών παραστάσεων έγινε με την χρήση της βιβλιοθήκης matplotlib της python. Όλα τα scripts, τα αποτελέσματα τύπου .txt και τα αρχεία python έχουν συμπεριληφθεί στο turnin που κάναμε στον φάκελο Files.

Σημαντικό είναι εδώ να αναφέρουμε ότι όταν τρέχουμε writeread με 1 thread, χρησιμοποιείτε 1 thread για το write και 1 thread για το read, δηλαδή γίνεται και σε αυτήν την περίπτωση ταυτοχρονισμός(αναφορά στο δεύτερο γράφημα, στην γραφική παράσταση με μπλε χρώμα). Στην καθαρή όμως σύγκριση με 1 thread για το read, 1 thread για το write και 4 thread για το writeread, ο πολυνηματισμός λειτουργεί καλύτερα(αναφορά στο πρώτο γράφημα, που συγκρίνουμε την παλιά λειτουργία που είναι μη-πολυνηματική με την καινούργια μας υλοποίηση, αυτήν με την πολυνηματική λειτουργία). Ετσι όλες οι συγκρίσεις παρακάτω που θα γίνονται writeread 1 είναι για 2 thread 1 για το read και 1 για το write.

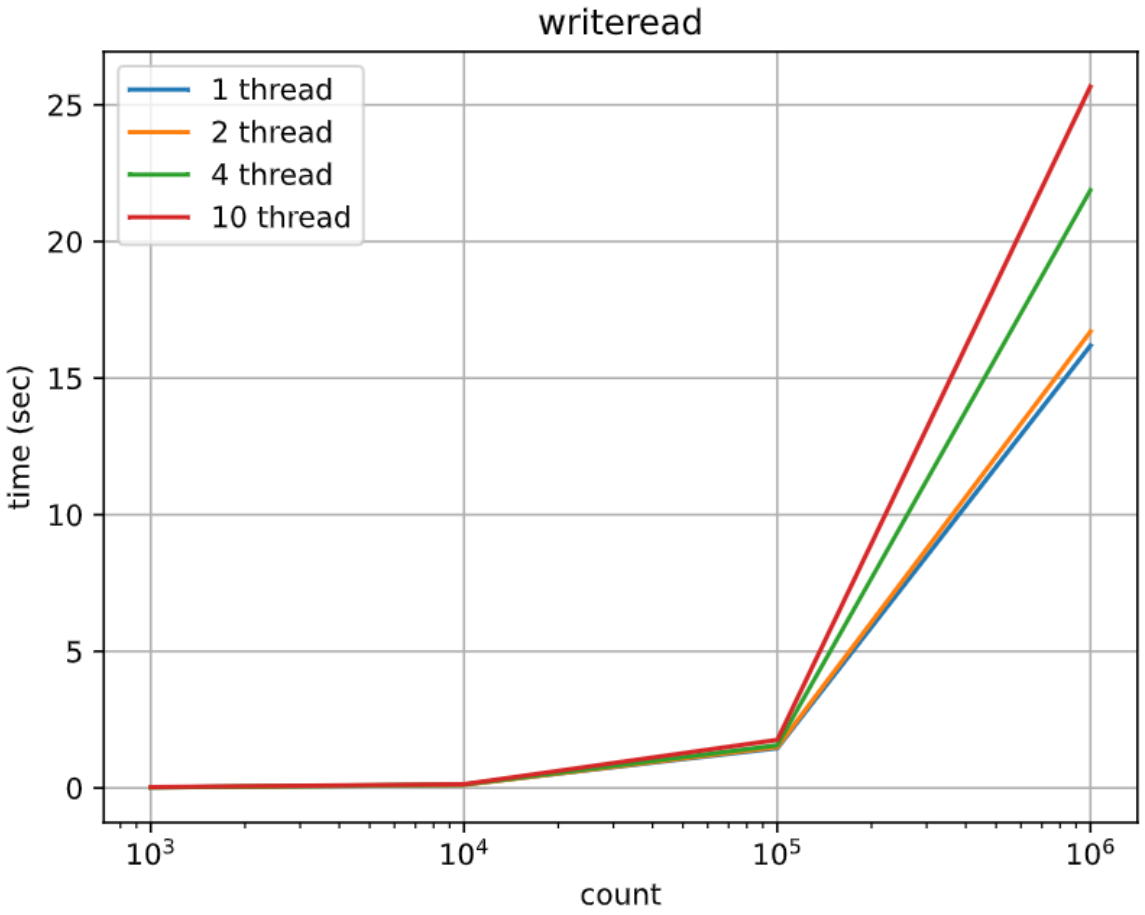
Βασική προϋπόθεση είναι ότι όλες οι μετρήσεις έγιναν στην εικονική μηχανή στην οποία δόθηκε 1 επεξεργαστικός πυρήνας. Στην επόμενη ενότητα εξετάζουμε τα αποτελέσματα και με την χρήση περισσότερων πυρήνων, όπου εκεί φαίνεται καθαρά ο ταυτοχρονισμός, απλά λόγω πιο σταθερών αποτελεσμάτων επιλέξαμε σε αυτήν την φάση, στην κατασκευή των γραφημάτων, να εργαστούμε με έναν πυρήνα. Παρατηρίσαμε σπάνια(περίπου 1 φορά στις 5 κλίσεις όταν το τρέχαμε με 4 πυρήνες) να έβγαζε μηνύματα τύπου bash error ή segmentation fault όταν τρέχαμε πολλές φορές και πολύ μεγάλο όγκο παραδειγμάτων, πράγμα που μας δυσκόλευε στις μετρήσεις με την τεχνική που επιλέξαμε να ακολουθήσουμε και δεν θα έβγαιναν τα βέλτιστα αποτελέσματα. Υποθέτουμε, λόγω της συγκεκριμένης εργασίας όπου το read εξαρτάται άμεσα με το write, σε συνδυασμό με πιθανότατα την όχι τόσο βαθιά εύρεση της κρίσιμης περιοχής, είχαμε ορισμένες περιπτώσεις που προκαλούταν σύγχυση ανάμεσα σε αυτές τις δύο λειτουργίες.

Μη Πολυνηματισμός VS Πολυνηματισμός



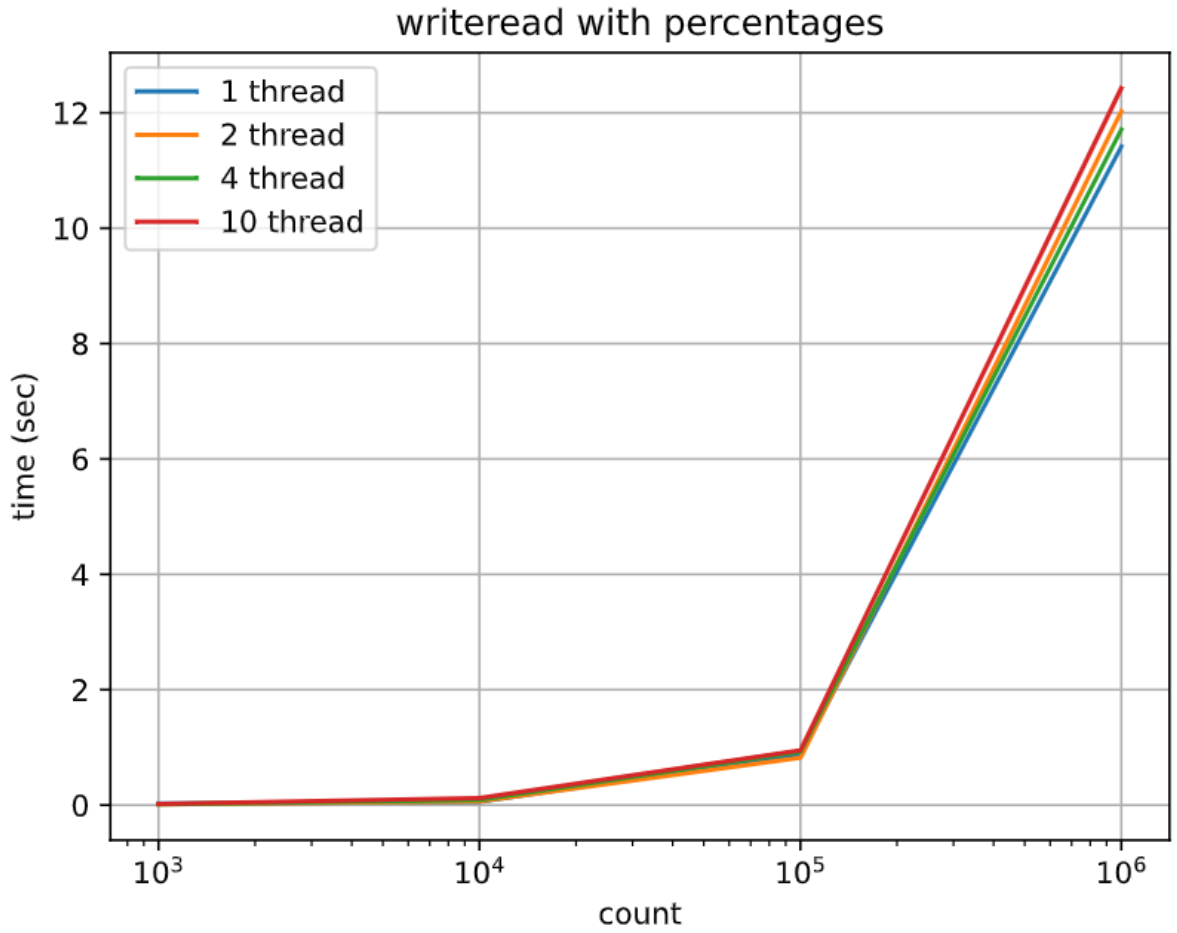
Στο παραπάνω γράφημα βλέπουμε το χρόνο που χρειάζονται τα read και τα write με 1 thread(απλή κλίση, μη-πολυνηματική) και τα writeread με 4 thread(νέα κλίση, πολυνηματική). Ο χρόνος, μετά την πρόσθεση του χρόνου write με του χρόνου read (το αθροισμα βγαίνει=18.929 στο 1000000), είναι μεγαλύτερος από τα writeread (17.922 στο 1000000). Αυτό είναι μία ακόμα ένδειξη ότι ο πολυνηματισμός λειτουργεί σωστά όπως αναφέρθηκε και νωρίτερα. Στην ουσία η μεμονωμένη κλίση του write και του read, είναι λες και τρέχουμε ένα μόνο νήμα, το ίδιο νήμα που τρέχει όλη την συγκεκριμένη εκτέλεση, άρα είναι μη-πολυνηματική κλίση. Ενώ η κλίση writeread είναι η καινούργια μας πολυνηματική υλοποίηση.

Πολυνηματισμός writeread



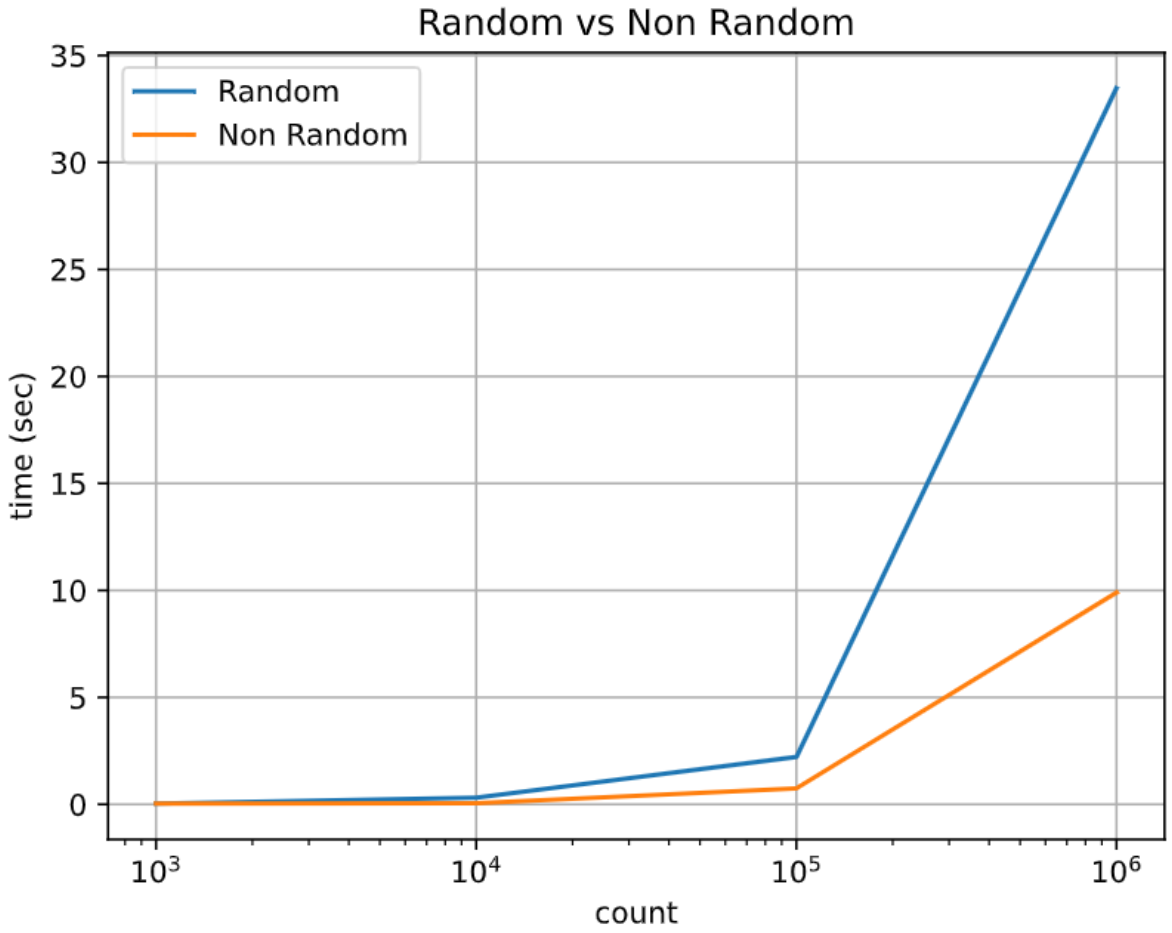
Στο παραπάνω γράφημα βλέπουμε τον αριθμό των count που κάνουν οι εκτελέσεις του προγράμματος με διαφορετικό αριθμό νημάτων. Παρατηρούμε ότι η αύξηση των νημάτων δεν βελτιώνει την ταχύτητα του προγράμματος και υπάρχουν κάποιοι λόγοι για αυτό. Αν και ο πολυνηματισμός λειτουργεί σωστά όπως είδαμε και παραπάνω, είναι υλοποιημένος σε ένα σημείο ψηλά στον κώδικα. Έτσι οι χρόνοι που περιμένουν τα νήματα είναι αρκετά μεγάλοι και ο ταυτοχρονισμός δεν μπορεί να ισοσταθμίσει αυτούς τους χρόνους και συνεπώς να μειώσει το χρόνο που τρέχει το πρόγραμμα. Επειδή τρέχουμε το πρόγραμμα σε υπολογιστή με ένα πυρήνα, τα threads μοιράζονται τους επεξεργαστικούς πόρους. Επομένως, παρότι αυξάνουμε τον αριθμό των νημάτων, παρατηρούμε ότι δεν βελτιώνεται η απόδοση, αλλά μπορεί μέχρι και να την χαλάσει, λόγω της αύξησης του χρόνου από τη δημιουργία των threads και από το συγχρονισμό τους(overhead time). Σε γενικές γραμμές, θα λειτουργούσε καλύτερα ο πολυνηματισμός με την χρήση πολλών πυρήνων. Σε αυτή την περίπτωση, το κάθε thread θα μπορούσε να τρέξει σε διαφορετικό πυρήνα, άρα με αυτόν τον τρόπο ο ταυτοχρονισμός θα επέφερε μείωση του χρόνου.

Πολυνηματισμός writeread με διαχωρισμό αιτήσεων 80-20



Στο παραπάνω γράφημα βλέπουμε τα ποσοστά των writeread ανάλογα τα threads. Τα ποσοστά είναι 80% write 20% read. Από αυτό το γράφημα βλέπουμε πάλι ότι τα πολλά threads επιβραδύνουν το πρόγραμμα. Το αξιοσημείωτο εδώ είναι ότι οι τιμές που παίρνουν είναι πιο κοντά από ότι κάποιος θα περίμενε βλέποντας τα παραπάνω γραφήματα. Αυτό συμβαίνει επειδή το πρόγραμμα τρέχει για πολύ λιγότερα read(αφού έχουμε πάρει το 20% από τις αιτήσεις) τα οποία τρέχουν με διαφορετικό αριθμό threads (1,2,4,10) κάτι που δε γίνεται με τα write. Τα write τρέχουν μόνιμα με 1 thread. Για αυτό το λόγο οι γραφικές παραστάσεις είναι τόσο κοντά.

Τυχαίες VS μη-Τυχαίες Τιμές Κλειδιών



Βλέπουμε στο παραπάνω γράφημα ότι όταν έχουμε μη τυχαίες τιμές κλειδιά το πρόγραμμα τρέχει σε καλύτερο χρόνο από όταν έχουμε τυχαίες τιμές. Αυτό πιστεύουμε πως γίνεται για δύο λόγους. Ο πρώτος είναι πως για να βρούμε κάθε φορά μία random τιμή πρέπει να τρέξει η συνάρτηση "_random_key" που καθυστερεί την εκτέλεση του προγράμματος. Ο δεύτερος λόγος είναι πως οι τιμές επειδή δεν έχουν κάποια σειρά είναι πιο δύσκολο να βρεθούν από το πρόγραμμα. Αυτό συμβαίνει καθώς βάζοντας τυχαίες τιμές αυξάνεται το μήκος του πεδίου ορισμού των κλειδιών που σημαίνει πως αλγόριθμοι ταξινόμησης και συνεπώς αλγόριθμοι αναζήτησης καθυστερούν καθώς οι αλγόριθμοι ταξινόμησης πετυχαίνουν καλύτερες πολυπλοκότητας με μικρά και γνωστά πεδία ορισμού.

Παραδείγματα Εκτέλεσης στο Τερματικό

Πρώτα Βήματα

Καλό είναι, πριν τρέξουμε το πρόγραμμά μας, να εκτελέσουμε 2 εντολές στο τερματικό, την make clean και έπειτα την make all.

```
myy601@myy601lab1:~/2023/kiwi/kiwi-source$ make clean
cd engine && make clean
make[1]: Entering directory '/home/myy601/2023/kiwi/kiwi-source/engine'
rm -rf *.o libindexer.a
make[1]: Leaving directory '/home/myy601/2023/kiwi/kiwi-source/engine'
cd bench && make clean
make[1]: Entering directory '/home/myy601/2023/kiwi/kiwi-source/bench'
rm -f kiwi-bench
rm -rf testdb
make[1]: Leaving directory '/home/myy601/2023/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/2023/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/2023/kiwi/kiwi-source/engine'
    CC db.o
db.c: In function ‘db_open_ex’:
db.c:32:51: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
    printf("........I AM WAITIMG(CurrentThreadId: %d)........\n", pthread_self()); //an tipvthei auto epivevaivnei oti stin arxh exei klhthei
to nhma read
                                                 ~^              ~~~~~~~~~~~~~~
                                                 %ld
db.c:35:50: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("........I AM RUNNING(CurrentThreadId: %d)........\n", pthread_self());
                                                ~^              ~~~~~~~~~~~~~~
                                                %ld
db.c:49:47: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("........I AM HERE(CurrentThreadId: %d)........\n", pthread_self());
                                               ~^              ~~~~~~~~~~~~~~
                                               %ld

db.c: In function ‘db_close’:
db.c:65:33: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("____CurrentThreadId: %d\n", pthread_self());
                               ~^     ~~~~~~~~~~~~~~
                               %ld
db.c:71:33: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("____CurrentThreadId: %d\n", pthread_self());
                               ~^     ~~~~~~~~~~~~~~
                               %ld
db.c:91:46: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("........I FINISH(CurrentThreadId: %d)........\n", pthread_self());
                                              ~^              ~~~~~~~~~~~~~~
                                              %ld
```

```
    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/myy601/2023/kiwi/kiwi-source/engine'
cd bench && make all
make[1]: Entering directory '/home/myy601/2023/kiwi/kiwi-source/bench'
gcc -g -ggdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -lpthread -lsnappy -o ki
wi-bench
bench.c: In function ‘main’:
bench.c:208:34: warning: passing argument 3 of ‘pthread_create’ from incompatible pointer type [-Wincompatible-pointer-types]
    pthread_create(&tid[0], NULL, _write_test, (void *) &thread_args_write);
                                  ^~~~~~~~~~~
In file included from bench.c:1:
/usr/include/pthread.h:236:15: note: expected ‘void * (*)(void *)’ but argument is of type ‘void (*)(void *)’
       void *(*__start_routine) (void *),
       ~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~
bench.c:214:36: warning: passing argument 3 of ‘pthread_create’ from incompatible pointer type [-Wincompatible-pointer-types]
    pthread_create(&tid[i+1], NULL, _read_test, (void *) &thread_args_read); //dhmiourgoume to nhma tou read
                                    ^~~~~~~~~~
In file included from bench.c:1:
/usr/include/pthread.h:236:15: note: expected ‘void * (*)(void *)’ but argument is of type ‘void (*)(void *)’
       void *(*__start_routine) (void *),
       ~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~
bench.c:224:29: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
  printf("___WriteThread-1___%d\n", tid[0]);
                            ~^     ~~~~~~
                            %ld
bench.c:226:31: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘pthread_t’ {aka ‘long unsigned int’} [-Wformat=]
   printf("___ReadThread-%d___%d\n", i, tid[i]);
                              ~^         ~~~~~~
                              %ld
```

```
kiwi.c: In function '_write_test':
kiwi.c:19:15: warning: unused variable 'd' [-Wunused-variable]
  struct data *d = (struct data *) arg;
              ^
kiwi.c: In function '_read_test':
kiwi.c:76:15: warning: unused variable 'd' [-Wunused-variable]
  struct data *d = (struct data *) arg;
              ^
make[1]: Leaving directory '/home/myy601/2023/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/2023/kiwi/kiwi-source$
```

Επίσης, βλέπουμε ότι περνάει από compile, αφού δεν μας εμφανίζει κάποιο μήνυμα λάθους, παρά μόνο ορισμένα warnings που έχουν ληφθεί υπόψη, αλλά έχουν αγνοηθεί αφού δεν προκαλούν κάποιο πρόβλημα.

Τρόπος Κλίσης στο Τερματικό

Για την σωστή κλίση οποιασδήποτε λειτουργίας θέλουμε, χρήσιμο είναι να συμβουλευτούμε τον παρακάτω πίνακα. Ξεκινώντας την κλίση με το *./kiwi-bench* και συνεχίζοντας με όποιο επιτρεπτό συνδυασμό θέλουμε από αυτόν τον πίνακα. Επίσης οι θέσεις αυτού του πίνακα μας βοήθησαν στην συγγραφή του κώδικα που περιέχεται στο αρχείο bench.c.

			μπορεί να παραληφθεί		
1	2	3	(4	5)	(argc-1)
write read	count [π.χ. 100.000 αιτήσεις]	-	-	-	(-/no)RandomKeys
writeread		num_of_threads [π.χ. 4 νήματα read]	(ποσοστό για τις αιτήσεις του write)	(ποσοστό για τις αιτήσεις του read)	
			*πρέπει ο χρήστης να ελέγχει μόνος του αν το άθροισμα του ζεύγους ποσοστών είναι 100		

Έλεγχος Ορθότητας του Πολυνηματισμού

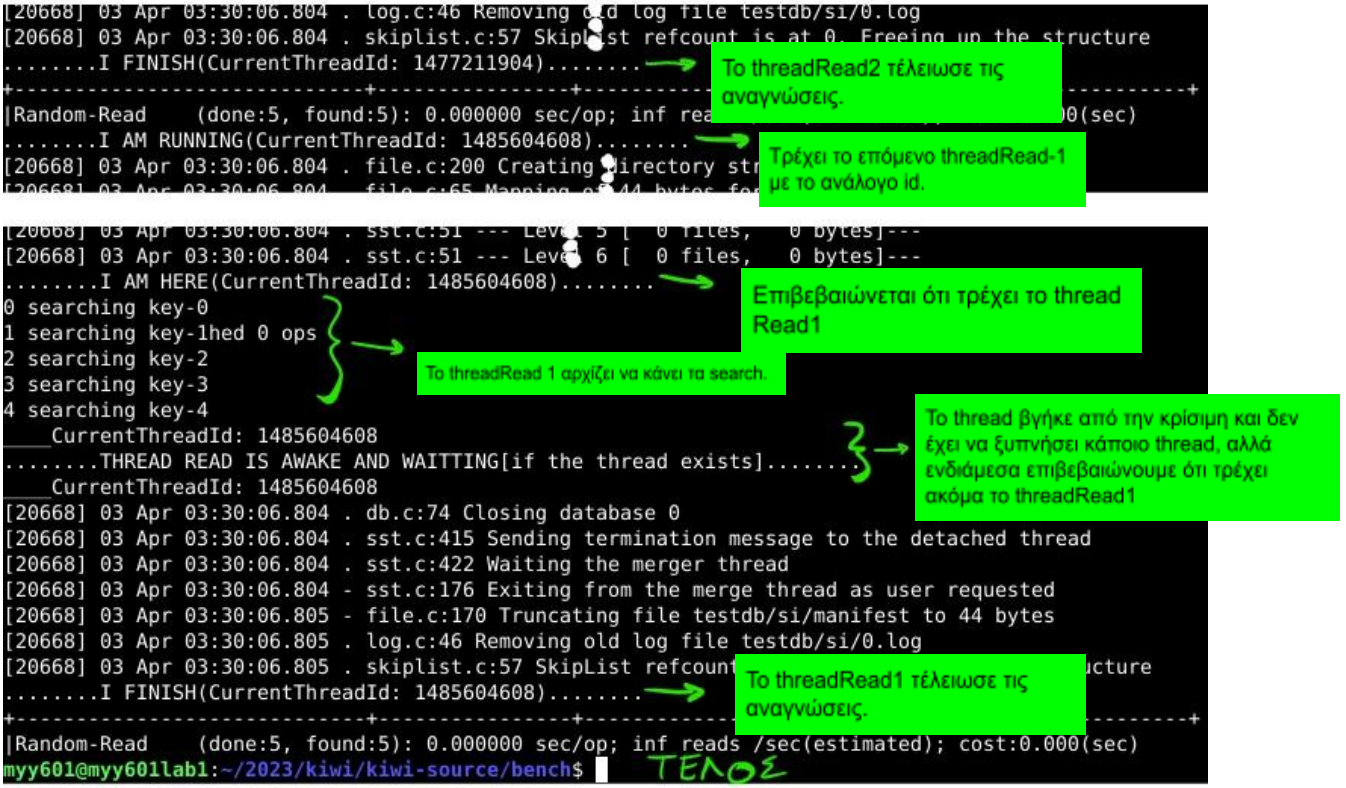
Για να ελέγξουμε την ορθή λειτουργία του πολυνηματισμού αρκεί να μπορούμε να διακρίνουμε την συμπεριφορά των νημάτων σε κάθε περίπτωση. Αυτό το επιτυνχάνουμε προσθέτοντας σε κρίσιμα σημεία του κώδικα, στο αρχείο db.c, διάφορες εκτυπώσεις που αφορούν την κατάσταση και τα id των νημάτων. Στις παρακάτω φωτογραφίες, σε συνδυασμό με τους σχολιασμούς, αναπαριστώνται μερικά παραδείγματα για να επιβεβαιώσουν τα παραπάνω.

Το πρώτο παράδειγμα θα είναι μία κλίση της συνάρτησης writeread, όπου είναι υπεύθυνη για τον πολυνηματισμό των συναρτήσεων write και read, με 2 νήματα read, χωρίς ποσοστά και με μη τυχαίες τιμές για τα κλειδιά. Η βασική διαφορά που θα έχει αυτό το παράδειγμα με το επόμενο, είναι ότι το συγκεκριμένο εκτελέστηκε σε 1 επεξεργαστικό πυρήνα.

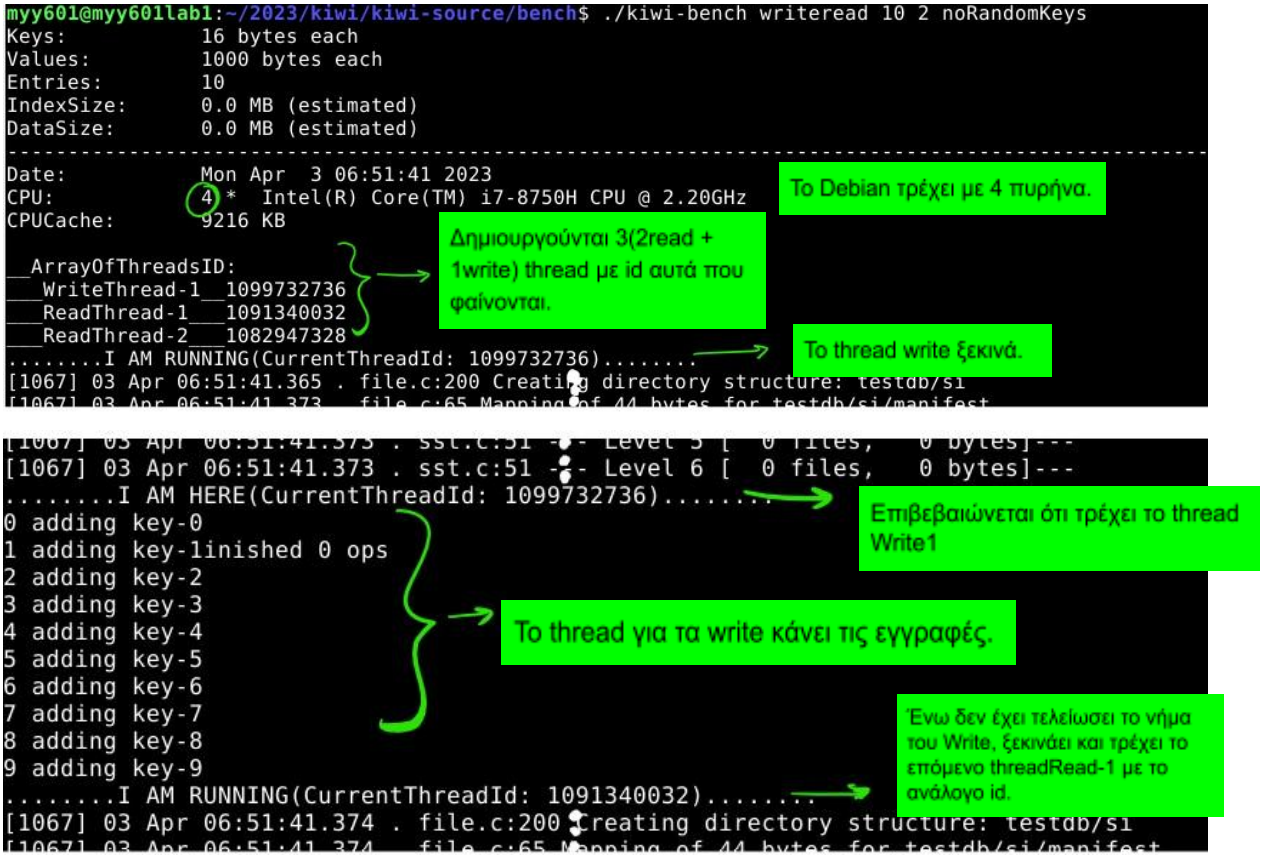

myy601@myy601lab1:~/2023/kiwi/kiwi-source/bench$ ./kiwi-bench writeread 10 2 noRandomKeys
Keys: 16 bytes each
Values: 1000 bytes each
Entries: 10
IndexSize: 0.0 MB (estimated)
DataSize: 0.0 MB (estimated)
Date: Mon Apr 3 03:30:06 2023
CPU: 1 * Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
CPUCache: 9216 KB
Το Debian τρέχει με 1 πυρήνα.
__ArrayOfThreadsID:
___WriteThread-1__1493997312
___ReadThread-1___1485604608
___ReadThread-2___1477211904
Δημιουργούνται 3 thread με id αυτά που φαίνονται.
.....I AM WAITIMG(CurrentThreadId: 1477211904).....
.....I AM WAITIMG(CurrentThreadId: 1485604608).....
Τα thread για το read περιμένουν.
.....I AM RUNNING(CurrentThreadId: 1493997312).....
Το thread write ξεκινά.
[20668] 03 Apr 03:30:06.785 . file.c:200 Creating directory stru
[20668] 03 Apr 03:30:06.785 - file.c:211 -> Creating testdb
[20668] 03 Apr 03:30:06.791 - file.c:211 -> Creating testdb/si
[20668] 03 Apr 03:30:06.791 . sst.c:283 Manifest file not present
.....I AM HERE(CurrentThreadId: 1493997312).....
Επιβεβαιώνεται ότι τρέχει το thread write
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
Το thread για τα write κάνει τις εγγραφές.
___CurrentThreadId: 1493997312
.....THREAD READ IS AWAKE AND WAITTING[if the thread exists].....
Το thread βγήκε από την κρίσιμη περιοχή και μπορεί να ξυπνήσει το επόμενο thread
___CurrentThreadId: 1493997312
[20668] 03 Apr 03:30:06.792 . db.c:74 Closing database 10
[20668] 03 Apr 03:30:06.792 . sst.c:595 N sst merge the REFCOUNT IS at 2

[20668] 03 Apr 03:30:06.799 - file.c:170 Truncating file testdb/si/manifest to 44 bytes
[20668] 03 Apr 03:30:06.800 . log.c:46 Removing old log file
.....I FINISH(CurrentThreadId: 1493997312).....
Το thread τέλειωσε τις εγγραφές.
|Random-Write (done:10): 0.000000 sec/op; inf writes/sec(estim
.....I AM RUNNING(CurrentThreadId: 1477211904).....
Τρέχει το επόμενο threadRead-2 με το ανάλογο id.
[20668] 03 Apr 03:30:06.800 . file.c:200 Creating directory stru

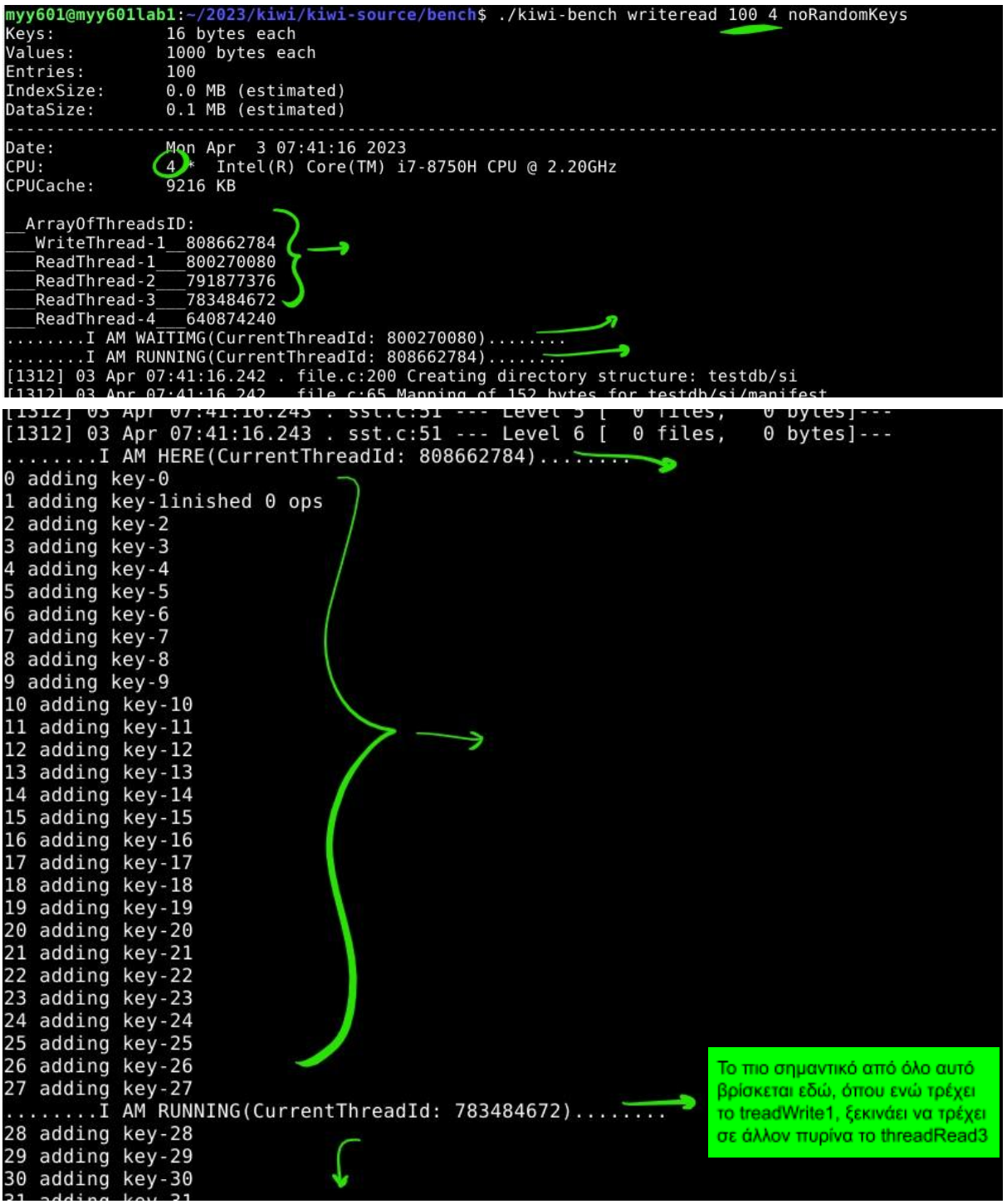
[20668] 03 Apr 03:30:06.800 . sst.c:51 - Level 5 [0 files, 0 bytes]---
[20668] 03 Apr 03:30:06.800 . sst.c:51 - Level 6 [0 files
.....I AM HERE(CurrentThreadId: 1477211904).....
Επιβεβαιώνεται ότι τρέχει το thread Read2
0 searching key-0
1 searching key-1hed 0 ops
2 searching key-2
3 searching key-3
4 searching key-4
Το threadRead 2 αρχίζει να κάνει τα search.
___CurrentThreadId: 1477211904
.....THREAD READ IS AWAKE AND WAITTING[if the thread exists].....
Το thread βγήκε από την κρίσιμη περιοχή και μπορεί να ξυπνήσει το threadRead1, αλλά ενδιάμεσα επιβεβαιώνουμε ότι τρέχει ακόμα το threadRead2
___CurrentThreadId: 1477211904
[20668] 03 Apr 03:30:06.800 . db.c:74 Closing database 0
[20668] 03 Apr 03:30:06.800 . sst.c:415 Sending termination message to the detached thread



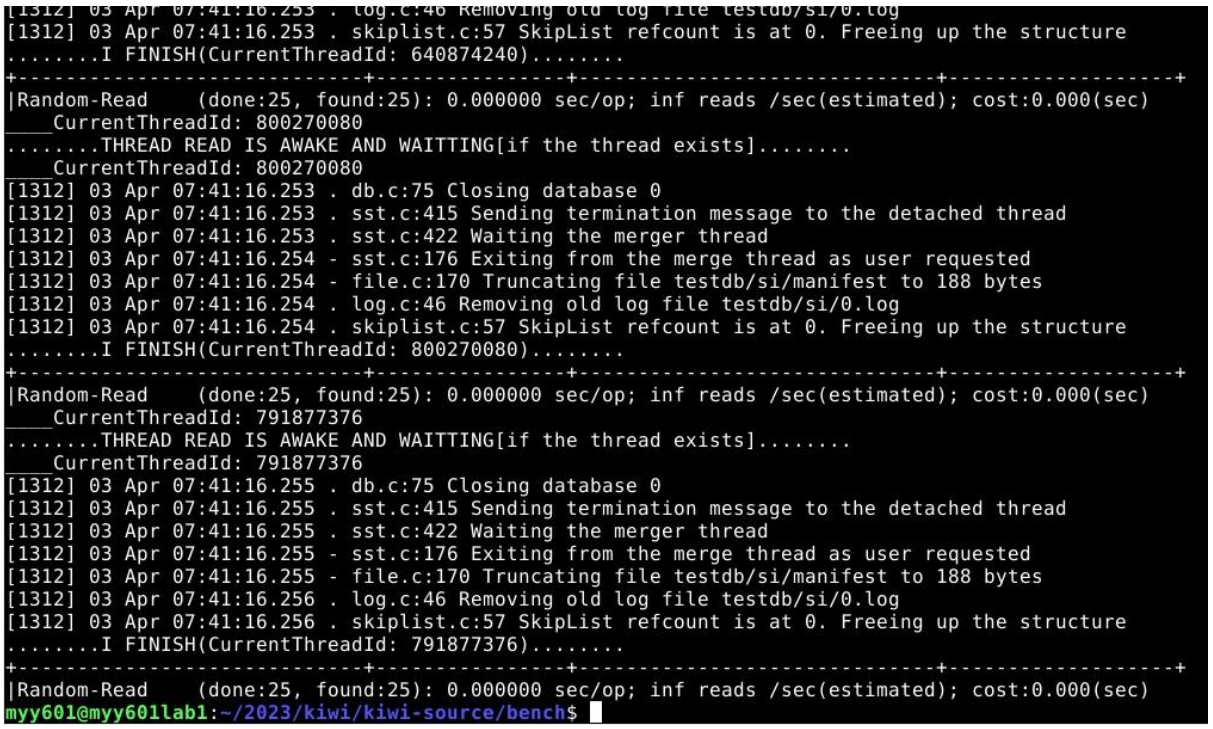
Τώρα, στο δεύτερο παράδειγμα, θα δούμε το ίδιο ακριβώς παράδειγμα με το παραπάνω, όμως θα το εκτελέσουμε σε ένα σύστημα με 4 πυρήνες, για να γίνει το φαινόμενο του ταυτοχρονισμού ευδιάκριτο.



Κάνουμε ένα τελευταίο, λίγο πιό σύντομο παράδειγμα, με μεγαλύτερο αριθμό αιτήσεων, προκειμένου να δούμε ακόμα πιο καθαρά την πραγματικά ταυτόχρονη λειτουργία των νημάτων. Γενικά στα βελάκια θα έπρεπε να μπουν παρόμοια πράγματα με τα απο πάνω.



... και μετά συνεχίζει με παρόμοιο τρόπο, ώσπου μετά από λίγο, αφού τελειώσει όλες τις εγγραφές και αναγνώσεις που πρέπει, τερματίζει.


[1312] 03 Apr 07:41:16.253 . log.c:46 Removing old log file testdb/si/0.log
[1312] 03 Apr 07:41:16.253 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
........I FINISH(CurrentThreadId: 640874240).........
+----------------------------+---------------+----------------------------+------------------+
|Random-Read   (done:25, found:25): 0.000000 sec/op; inf reads /sec(estimated); cost:0.000(sec)
____CurrentThreadId: 800270080
........THREAD READ IS AWAKE AND WAITTING[if the thread exists].........
____CurrentThreadId: 800270080
[1312] 03 Apr 07:41:16.253 . db.c:75 Closing database 0
[1312] 03 Apr 07:41:16.253 . sst.c:415 Sending termination message to the detached thread
[1312] 03 Apr 07:41:16.253 . sst.c:422 Waiting the merger thread
[1312] 03 Apr 07:41:16.254 - sst.c:176 Exiting from the merge thread as user requested
[1312] 03 Apr 07:41:16.254 - file.c:170 Truncating file testdb/si/manifest to 188 bytes
[1312] 03 Apr 07:41:16.254 . log.c:46 Removing old log file testdb/si/0.log
[1312] 03 Apr 07:41:16.254 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
........I FINISH(CurrentThreadId: 800270080).........
+----------------------------+---------------+----------------------------+------------------+
|Random-Read   (done:25, found:25): 0.000000 sec/op; inf reads /sec(estimated); cost:0.000(sec)
____CurrentThreadId: 791877376
........THREAD READ IS AWAKE AND WAITTING[if the thread exists].........
____CurrentThreadId: 791877376
[1312] 03 Apr 07:41:16.255 . db.c:75 Closing database 0
[1312] 03 Apr 07:41:16.255 . sst.c:415 Sending termination message to the detached thread
[1312] 03 Apr 07:41:16.255 . sst.c:422 Waiting the merger thread
[1312] 03 Apr 07:41:16.255 - sst.c:176 Exiting from the merge thread as user requested
[1312] 03 Apr 07:41:16.255 - file.c:170 Truncating file testdb/si/manifest to 188 bytes
[1312] 03 Apr 07:41:16.256 . log.c:46 Removing old log file testdb/si/0.log
[1312] 03 Apr 07:41:16.256 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
........I FINISH(CurrentThreadId: 791877376).........
+----------------------------+---------------+----------------------------+------------------+
|Random-Read   (done:25, found:25): 0.000000 sec/op; inf reads /sec(estimated); cost:0.000(sec)
myy601@myy601lab1:~/2023/kiwi/kiwi-source/bench$