Λειτουργικά Συστήματα

Εργαστήριο 1

Καραγιαννίδης Χρήστος , Α.Μ : 4375

Χριστόπουλος Κων/νος , Α.Μ : 4527

bench.c :

Οι αλλαγές που κάναμε σε αυτό το αρχείο είναι :

```
if (argc < 3) {
    fprintf(stderr,"Usage: db-bench <write | read | readwrite> <count> <thread number>\n");  //added readwrite
    exit(1);
}
```

Αλλάξαμε την “print” ώστε να δείχνει τον σωστό ορισμό του προγράμματος.

```
} else if (strcmp(argv[1], "readwrite") == 0) {      //an dwthei readwrite ws orisma mpainei edw
    int r = 0;                                       //arxikopoiisi tou r (an r=1 exoume random values sta read / write)

    count = atoi(argv[2]);                           //metatrepei to 3o orisma se int gia na aksiopoiithei stin ylopoiisi
    _print_header(count);                            //oi 2 print xrisimopoiountai gia na moiazei i readwrite me tis read kai write se emfanisi sto termatiko
    _print_environment();

    if (argc == 4){
        r = 0;
        nhmata = atoi(argv[3]);                      //metatrepei to 4o orisma se int
    }
    if (argc == 5){
        r=0;
        nhmata = atoi(argv[3]);                      //metatrepei to 4o orisma se int ->nhmata
        percentage = atoi(argv[4]);                  //kai to 5o se int ->percentage
    }

    _readwrite_test(count, r, nhmata, percentage);          //kalei tin readwrite
```

Προσθέσαμε μια επιπλέον “else if{}” για την περίπτωση της read/write. Συγκεκριμένα αν το πρώτο όρισμα που δώσει ο χρήστης είναι της μορφής “readwrite” και το πρόγραμμα μπαίνει σε αυτήν την “else if{}” και μετά αναλόγως τα ορίσματα που θα δώσει επιπλέον αρχικοποιούνται οι μεταβλητές :

1.“count”(= αναφέρεται στα operations που θα γίνουν , reads/writes)

2. “nhmata”(= αναφέρεται στα threads που θα εκτελεστούν)

3. “percentage”(= αναφέρεται στο ποσοστό των reads που θα εκτελεστούν. Ωστόσο δεν αξιοποιήθηκε παρακάτω στην υλοποίηση του προγράμματος.)

Όλες οι “atoi()” χρησιμοποιούνται για να μετατρέψουν τα ορίσματα σε “int” που αργότερα θα χρησιμοποιηθούν στην υλοποίηση . Τέλος, καλείται η συνάρτηση “readwrite_test()”.

bench.h:

```
17  typedef struct data{
18      long int count;
19      int r;
20      int nhmata;
21      DB* db;
22      int found;
23  }Data;
24
25  pthread_mutex_t timeLock;
```

Στο bench.h δηλώσαμε μία struct με το όνομα Data , η οποία χρησιμοποιείται ως είσοδος για την συνάρτηση που καλεί η “pthread_create()” στο αρχείο kiwi.c . Επιπλέον δηλώνουμε ένα mutex με το όνομα timeLock, το οποίο και αυτό θα χρησιμοποιηθεί στο αρχείο kiwi.c .

kiwi.c:

```
8   #define DATAS ("testdb")
9   double Time_previous , Total_cost;  //Time_previous : global metablhth pou isoutai me thn xroniki stigmi pou teleiwse to teleftaio nhma
10
11  void _write_test(long int count, int r)
12  {
63  }
64
65  void _read_test(long int count, int r)
66  {
117 }
118
119 void* _read_test1(void * arg)
120 {
191 }
192
193 void *_write_test1(void *arg)
194 {
257 }
258
259 void _readwrite_test(long int count, int r, int nhmata , int percentage)
260 {
261     pthread_t tid[nhmata];                                  //dilwseis nhmatwn analogws me to posa zitisei o xristis
262     pthread_t threadid[2];                                  //2 epipleon threads ta opoia tha eksigisoume parakatw
263     Data thread_args;                                       //dilwsi mias metavlitis me to onoma "thread_args" typou Data (pou einai ena struct stin bench.h)
264     int i;                                                  //dilwsi metavlitis gia na tin xrisimopoiisoume stis for(;;)
265     if (nhmata>count){                                      //an ta nhmata einai perissotera apo ta counts prokalei segmentation fault
266         nhmata = count;                                     //opote valame na ginontai tosa nhmata oses kai oi prakseis pou xreiazontai
267     }                                                       //px. 10 ops kai 100 nhmata => 10 nhmata pou tha kanoun apo 1 op (kai ta alla 90 den tha dimiourgithoun kan)
268
269     thread_args.count = count/nhmata;                       //arxikopoiisi twn pediwn tou thread_args
270     thread_args.r = r;
271     thread_args.nhmata = nhmata;
272     thread_args.db = db_open(DATAS);
273     thread_args.found = 0;                                  //i thread_args.found den aksiopoieitai apo tis write wstoso einai aparaititi gia tis read
274
275     Time_previous  = get_ustime_sec();                      //time_previous pairnei ws timi tin enarksi tis readwrite_test
```

Αρχικά δηλώνουμε 2 global μεταβλητές με ονόματα :

1. Total_cost : όπου είναι είναι το συνολικό κόστος του προγράμματος σε seconds.
2. Time_previous : όπου ισούται με την χρονική στιγμή που τελείωσε το τελευταίο νήμα . Αυτή η μεταβλητή αρχικοποιείται στην γραμμή “275” και ισούται με την στιγμή που ξεκίνησε να εκτελείται η _readwrite_test().

Στην συνέχεια αυτή η μεταβλητή χρησιμοποιείται στο τέλος των συναρτήσεων _read_test1() και _write_test1().(πιο αναλυτικά στην σελίδα 5) .

_readwrite_test() : Αρχικά δηλώνουμε τα νήματα σε έναν πίνακα που ισούται με τον αριθμό των νημάτων που έχει δώσει σαν είσοδο ο χρήστης στην bench.c . Στην γραμμή "262" δηλώνουμε 2 επιπλέον νήματα τα οποία θα τα χρησιμοποιήσουμε λίγο παρακάτω. Βάλαμε μια "if{}" η οποία ελέγχει αν ο αριθμός των νημάτων που δόθηκαν από τον χρήστη είναι μεγαλύτερος από τον αριθμό των operations. Στην συγκεκριμένη περίπτωση ο αριθμός των threads γίνεται ίσος με τον αριθμό των operations, άρα κάθε νήμα εκτελεί μια operation (τα επιπλέον νήματα δεν δημιουργούνται ποτέ). Στις γραμμές "269-273" αρχικοποιούνται τα πεδία του thread_args τα οποία θα δοθούν ως ορίσματα στις συναρτήσεις που καλούν οι _pthread_create().

```
276     if(nhmata == 1){
277         pthread_create(&tid[0], NULL, _write_test1, (void *) &thread_args);        /* dimourgeitai 1 nhma kai ektelei tin write_test1*/
278         pthread_detach(tid[0]);                                                    /* apodesmevetai to nhma me tin detach*/
279         db_close(thread_args.db);                                                  /* kleinei to db*/
280         printf("_write_test1() finished...\n");                                    /* 2 print gia na kseroume oti teleiwse to 1 kai ksekinise to allo*/
281
282         printf("Preparing for _read_test1()...\n");
283         thread_args.db = db_open(DATAS);                                           /* anoigei to db*/
284         pthread_create(&tid[0], NULL, _read_test1, (void *) &thread_args);         /* kaleitai i read_test me to idio thread*/
285         pthread_join(tid[0], NULL);                                                /* ginetai join to nhma*/
```

Αν τα νήματα που δόθηκαν από τον χρήστη είναι ίσο με 1 , μπαίνουμε σε αυτή την "if{}" , η οποία δημιουργεί ένα thread που καλεί την συνάρτηση _write_test1() . 'Επειτα αποδεσμεύει το νήμα κλείνει το database , κανει printf() ότι τελείωσε η _write_test1() και ότι ξεκινάει η _read_test1() , ανοίγει πάλι το database και στο ίδιο νήμα από πριν αναθέτει την εκτέλεση της _read_test1(). Τέλος κάνει join() το νήμα.

```
286     }else{
287         for(i = 0; i < nhmata; i ++){                                              /* Mexri o i na ftasei stin timi tou nhmata (ta opoia exoun dwthei ws input apo ton xristi) ektelese tin */
288             if (i%2 == 0){                                                         /* Pthread create enallaks mia gia na kalesei tin read_test1 kai mia gia na kalesei tin write_test1      */
289                 pthread_create(&tid[i], NULL, _write_test1, (void *) &thread_args);/* i enallagi epitigxanetai me to na checkaroume an to ypoloipo tis diairesis tou i me to 2 einai == 0   */
290             }                                                                      /* dld an o i einai artios ektaileitai i write_test1 enw perittos i read_test1                            */
291             else {
292                 pthread_create(&tid[i], NULL, _read_test1, (void *) &thread_args);
293             }
294         }
295
296
297         for(i = 0; i < nhmata; i ++){                                              /* kanoume join ta nhmata pou kaname create parapanw */
298             pthread_join(tid[i], NULL);
299         }
300
```

Στην περίπτωση που τα νήματα είναι περισσότερα του 1 εκτελείται η "else{}" . Συγκεκριμένα δημιουργούνται νήματα εναλλάξ τα οποία καλούν τις _read_test1() και _write_test1() αντίστοιχα , ανάλογα με το αν η τιμή του "i" είναι άρτιος ή περιττός αριθμός. Τέλος αποδεσμεύει τα νήματα με την χρήση της join().

```
if(count != thread_args.count*nhmata){                                  /* elegxoume an to count pou dwthike (diladi ta operations )*/
                                                                        /* einai iso me to posa nhmata anathikan     */
    printf("\n Effort to put/get the operations that were lost in the division of count/nhmata (when nhmata is odd)\n");   /* se kathe thread epi to posa threads dimiourgithikan  */
    thread_args.count = (count - thread_args.count*nhmata) / 2;         /* kai an den einai isa tote afti i if epixeirei */
    printf("\n Operations that were lost  = %ld\n", 2*thread_args.count);   /* na ektelesei ta operations pou den ektelestikan */
    for (i=0; i<2;i++){                                                 /* px. 100ops 3 threads 33ops per thread 33*3 = 99 != 100*/
        if (i%2 == 0){                                                  /* ara prospathei na kanei tin 1 xameni praksi*/
            pthread_create(&threadid[i], NULL, _write_test1, (void *) &thread_args);    /* edw xrisimopoiountai ta 2 parapanw threads */
                                                                        /*  pou dimiourgisame stin grammi 262 1 thread read kai 1 write*/
        }
        else{
            pthread_create(&threadid[i], NULL, _read_test1, (void *) &thread_args);
        }
    }


    for(i = 0; i < 2; i ++){
    pthread_join(threadid[i], NULL);                                    /* join ta 2 threads */
    }


}
```

Αυτή την “if{}” την δημιουργήσαμε προκειμένου να καλύψουμε τα operations τα οποία χάνονταν κατά την κλήση του προγράμματος με περιττό αριθμό threads και άρτιο operations(και το αντίστροφο) . Δηλαδή, αν καλέσουμε την _readwrite_test() για 100 operations με 27 νήματα στο κάθε νήμα ανατίθενται 3 operations και 3*27 = 81 , οπότε κανονικά θα χάνονταν 19 operations , ενώ με αυτή την “if{}” εκτελούνται οι 18 από τις 19(που σημαίνει ότι και αυτή δεν είναι τέλεια, αλλά ήταν ότι πιο κοντινό στον αρχικό αριθμό των operations μπορούσαμε να βρούμε).

Μέσα στην “if{}” υπολογίζουμε πόσα operations δεν εκτελέστηκαν και ανατίθενται από μισά σε κάθε ένα από τα 2 επιπλέον νήματα που δηλώσαμε στην γραμμή “262”.

```


    }
    db_close(thread_args.db);                                           /*kleinoume to db*/
    printf(LINE1);                                                      /* print grammi gia na ta diaxwrizoume*
    printf("|Random-Read/Write  (done:%ld, found:%d): %.6f sec/op; %.1f reads/writes /sec(estimated); Total cost:%.3f(sec) with %d threads\n",
        count, thread_args.found,
        (double)(Total_cost / count),                                   /*print final stats*/
        (double)(count / Total_cost),
        Total_cost, nhmata);

}
```

Εδώ κλείνουμε database και εκτυπώνουμε τα στατιστικά της απόδοσης λειτουργιών.

Αφήσαμε τις _read_test() και _write_test() και φτιάξαμε δικές μας _read_test1() και _write_test1() αντίστοιχα , οι οποίες έχουν διαφορετικά ορίσματα από τις προηγούμενες ,αλλά η φιλοσοφία παραμένει ίδια με μικροδιαφορές.

_write_test1() :

Όλες οι δηλώσεις - αρχικοποιήσεις των μεταβλητών και η “for{}” έχουν την ίδια λογική και η μόνη διαφορά είναι το timeLock που έχουμε προσθέσει και οι μεταβλητές Time_previous και Total_cost.

```
193  void *_write_test1(void *arg)          /* pairnei ws orismata ta args*/
194  {
195      Data *d = (Data *) arg;            /* xrisimopoiountai oles oi metavlites pou xrisimopoiountan kai stin original _write_test() */
196      int i;
197      double cost;
198      long long start,end;
199      Variant sk_w, sv_w;
200
201      char key_w[KSIZE + 1];
202      char val[VSIZE + 1];
203      char sbuf[1024];
204
205      memset(key_w, 0, KSIZE + 1);
206      memset(val, 0, VSIZE + 1);
207      memset(sbuf, 0, 1024);
208
209      pthread_mutex_init(&timeLock, NULL);                    /* kanoume initialize to timeLock pou orisame sto bench.h*/
210      pthread_mutex_lock(&timeLock);                          /* kanoume lock to mutex timeLock prokeimenou na kratisei ta statistika tou xronou */
211      printf(LINE);                                           /* pou tha apothikefsoume se ligo xwris na ta epireasoun ta alla nimata*/
212      printf("Entering function _write_test1() with <%ld> (thread id)\n", pthread_self());
213      start = get_ustime_sec();                               /* ta start/end/cost einai i metavlites gia ton xrono pou ipirxan idi*/
214
```

Αρχικοποιούμε ένα pointer d σε Data με τιμές πεδίων τα πεδία του arg που δόθηκαν . Κάνουμε initialize και lock το timeLock προκειμένου να προστατέψουμε την κρίσιμη περιοχή της “for{}” και τις κρίσιμες global μεταβλητές Total_cost και Time_previous.

```
216      for (i = 0; i < d->count; i++) {
217          if (d->r)
218              _random_key(key_w, KSIZE);
219          else                                                //epeidh r=0 mpainei sto else
220              snprintf(key_w, KSIZE, "key-%d", i);
221          fprintf(stderr, "%d adding %s\n", i, key_w);
222          snprintf(val, VSIZE, "val-%d", i);
223
224          sk_w.length = KSIZE;                                //arxikopoiei tis times tou key kai ths value ta opoia einai struct Variant(=buffer.h)
225          sk_w.mem = key_w;
226          sv_w.length = VSIZE;                                /*OLI I FOR EINAI IDIA ME TIN FOR TIS ARXIKIS write_test() ME EKSAIRESI TA DEDOMENA POU ERXONTAI APO TA ARGS */
227          sv_w.mem = val;                                     /*TIS SYNARTISIS TA OPOIA ALLAKSANE (px. i metavliti 'db' egine 'd->db' ) */
228
229          db_add(d->db, &sk_w, &sv_w);                        //prosBetei kleidi-timh sto database(=db)
230          if ((i % 10000) == 0) {
231              fprintf(stderr,"random write finished %d ops%30s\r",
232                      i,
233                      "");
234
235              fflush(stderr);                                 //oti teleiwse 10000 operation
236          }
237      }
```

Η “for{}” δεν έχει αλλάξει σχεδόν καθόλου εκτός από τα δεδομένα που έρχονται από τα arg όπως π.χ το count το οποίο έγινε d->count.

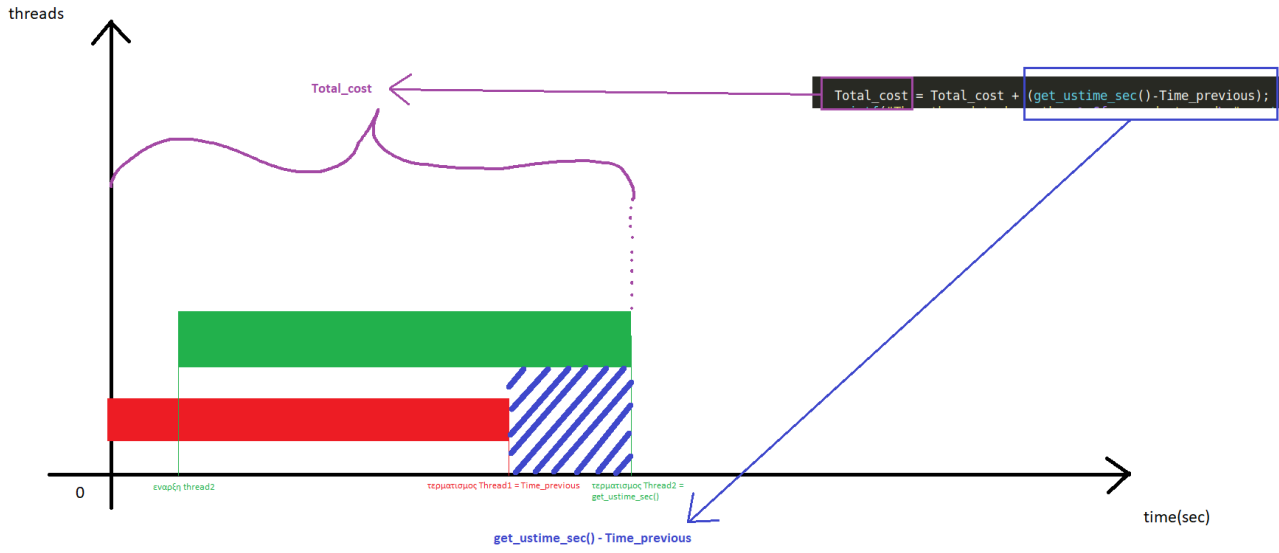
```
239
240      end = get_ustime_sec();
241      cost = end - start;                                 //ypologizei to topiko kostos(se xrono) pou xreiasthke gia na ginei auth h energeia
242
243      Total_cost = Total_cost + (get_ustime_sec()-Time_previous);          /* prosthetei sto total_cost tin sigkekrimeni xroniki stigmi 'meion'          */
244      printf("This thread took another %.6f seconds to end\n", get_ustime_sec()-Time_previous);   /* tin xroniki stigmi pou termatise to proigoumeno nima (an einai to lo nima    */
245      Time_previous = get_ustime_sec();                                    /* tote i previous exei tin timi stin opoia arxikopoiithike stin readwrite_test  */
246                                                                           /* epeita allazoume ws tin xroniki stigmi pou teleiwse to teleftaio nima to paron*/
247
248      printf(LINE);
249      printf("|Random-Write   (done:%ld): %.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec);\n"
250          ,d->count, (double)(cost / d->count)
251          ,(double)(d->count / cost)
252          ,cost);                                                          //edw tuponontai epimerous statistika tou nimatos
253      pthread_mutex_unlock(&timeLock);                                     /* unlock to timeLock gia na mporei na allaksei tin timi twn total cost kai  */
254      pthread_mutex_destroy(&timeLock);                                    /* time_previous ena allo nima kai meta deallocate to time lock*/
255
256      return 0;
257  }
```

Time_previous : κρατάει την χρονική στιγμή τερματισμού του προηγούμενου νήματος (αν δεν έχει τερματίσει κάποιο νήμα ακόμα , τότε η Time_previous έχει την τιμή της έναρξης της εκτέλεσης της _readwrite_test() η οποία αρχικοποιήθηκε στην γραμμή “275”).

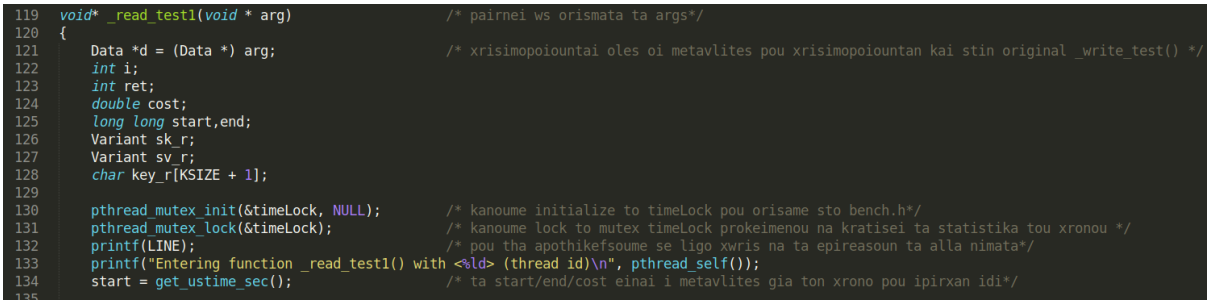
```
275     Time_previous  = get_ustime_sec();                    //time_previous pairnei ws timi tin enarksi tis readwrite_test
```

Total_cost : Υπολογίζει τον χρόνο εκτέλεσης όλης της _readwrite_test() από την στιγμή της έναρξής της μέχρι τον τερματισμό και του τελευταίου νήματός της. Συγκεκριμένα προσθέτει στον συνολικό χρόνο την χρονική διαφορά της συγκεκριμένης χρονικής στιγμής “μείον” την χρονική στιγμή που τελείωσε το προηγούμενο νήμα (βλ. σχήμα παρακάτω).

Στην γραμμή “245” θέτουμε ως τιμή τερματισμού του τελευταίου νήματος (Time_previous) την τρέχουσα χρονική στιγμή. Έπειτα κάνουμε unlock και destroy το mutex έτσι ώστε να μπορέσει να “λοκαριστεί” από κάποιο άλλο νήμα.



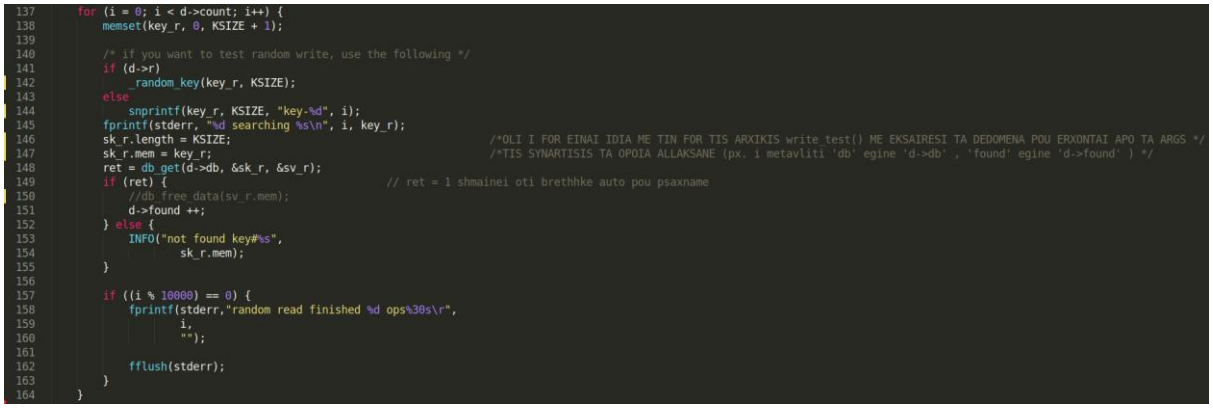
_read_test1() :



```
void* _read_test1(void * arg)                   /* pairnei ws orismata ta args*/
{
    Data *d = (Data *) arg;                     /* xrisimopoiountai oles oi metavlites pou xrisimopoiountan kai stin original _write_test() */
    int i;
    int ret;
    double cost;
    long long start,end;
    Variant sk_r;
    Variant sv_r;
    char key_r[KSIZE + 1];

    pthread_mutex_init(&timeLock, NULL);        /* kanoume initialize to timeLock pou orisame sto bench.h*/
    pthread_mutex_lock(&timeLock);              /* kanoume lock to mutex timeLock prokeimenou na kratisei ta statistika tou xronou */
    printf(LINE);                               /* pou tha apothikefsoume se ligo xwris na ta epireasoun ta alla nimata*/
    printf("Entering function _read_test1() with <%ld> (thread id)\n", pthread_self());
    start = get_ustime_sec();                   /* ta start/end/cost einai i metavlites gia ton xrono pou ipirxan idi*/
```

Ακριβώς όπως και στην _write_test1() , αρχικοποιούμε ένα pointer d σε Data με τιμές πεδίων τα πεδία του arg που δόθηκαν . Κάνουμε initialize και lock το timeLock προκειμένου να προστατέψουμε την κρίσιμη περιοχή της “for{}” και τις κρίσιμες global μεταβλητές Total_cost και Time_previous. Αρχικοποιούμε και κάποιες επιπλέον μεταβλητές που χρειάζεται η _read_test1() όπως είναι π.χ η ret.



```
    for (i = 0; i < d->count; i++) {
        memset(key_r, 0, KSIZE + 1);

        /* if you want to test random write, use the following */
        if (d->r)
            _random_key(key_r, KSIZE);
        else
            snprintf(key_r, KSIZE, "key-%d", i);
        fprintf(stderr, "%d searching %s\n", i, key_r);
        sk_r.length = KSIZE;                            /*OLI I FOR EINAI IDIA ME TIN FOR TIS ARXIKIS write_test() ME EKSAIRESI TA DEDOMENA POU ERXONTAI APO TA ARGS */
        sk_r.mem = key_r;                               /*TIS SYNARTISIS TA OPOIA ALLAKSANE (px. i metavliti 'db' egine 'd->db' , 'found' egine 'd->found' ) */
        ret = db_get(d->db, &sk_r, &sv_r);
        if (ret) {                              // ret = 1 shmainei oti brethhke auto pou psaxname
            //db_free_data(sv_r.mem);
            d->found ++;
        } else {
            INFO("not found key#%s",
                    sk_r.mem);
        }

        if ((i % 10000) == 0) {
            fprintf(stderr,"random read finished %d ops%30s\r",
                    i,
                    "");

            fflush(stderr);
        }
    }
```

Πάλι όπως στην _write_test1() , το μόνο που αλλάζει είναι ότι μεταβλητές όπως το found γίνονται d->found.

```
    end = get_ustime_sec();
    cost = end - start;                             //ypologizei to topiko kostos(se xrono) pou xreiasthke gia na ginei auth h energeia

    Total_cost = Total_cost + (get_ustime_sec()-Time_previous);          /* prosthetei sto total_cost tin sigkekrimeni xroniki stigmi 'meion'          */
    printf("This thread took another %.3fseconds to end\n", get_ustime_sec()-Time_previous);    /* tin xroniki stigmi pou termatise to proigoumeno nima (an einai to 1o nima     */
    Time_previous = get_ustime_sec();                                    /* tote i previous exei tin timi stin opoia arxikopoiithike stin readwrite_test  */
                                                                         /* epeita allazoume ws tin xroniki stigmi pou teleiwse to teleftaio nima to paron*/

    printf(LINE);
    printf("|Random-Read    (done:%ld, found:%d): %.6f sec/op; %.1f reads /sec(estimated); cost:%.3f(sec)\n",
        d->count, d->found,
        (double)(cost / d->count),
        (double)(d->count / cost),
        cost);                                                           //edw tuponontai epimerous statistika tou nimatos

    pthread_mutex_unlock(&timeLock);                                     //edw tuponontai epimerous statistika tou nimatos
    pthread_mutex_destroy(&timeLock);                                    /* unlock to timeLock gia na mporei na allaksei tin timi twn total cost kai  */
                                                                         /* time_previous ena allo nima kai meta deallocate to time lock*/
    return 0;
}
```

Τα Time_previous και Total_cost λειτουργούν με ακριβώς τον ίδιο τρόπο όπως στην _write_test1() και αλλάζουν τις ίδιες global μεταβλητές γι’ αυτό και χρησιμοποιούν το ίδιο mutex(timeLock) το οποίο γίνεται unlock και destroy στο τέλος για να χρησιμοποιηθεί από άλλο νήμα.

db.h:

Σε αυτό το αρχείο δηλώσαμε τα mutex που θα τα χρησιμοποιήσουμε στις db_add() και db_get() αντιστοίχως.

```
19
20  pthread_mutex_t put_locker;
21  pthread_mutex_t get_locker;
22
```

db.c:

```
11  pthread_cond_t cond = PTHREAD_COND_INITIALIZER;     /*arxikopoiisi tis metavlitis sinthikis cond*/
12  int condition_check = 1;                            /*arxikopoiisi tou condition check sto 1 wste to 1o nima pou tha ektelestei na min kollisei stin while*/
```

Αρχικοποιούμε ως global την συνθήκη cond και την μεταβλητή condition_check που θα χρησιμοποιηθούν στην db_add().

```
53  int db_add(DB* self, Variant* key, Variant* value)
54  {
55      int returnVariable;                         /*metavliti pou apothikevei tin timi pou epistrefei i memtable_add(self->memtable, key, value)*/
56      pthread_mutex_init(&put_locker, NULL);      /*prokeimenou na mpei mesa stin perioxi tou mutex*/
57      pthread_mutex_lock(&put_locker);            /*arxikopoiisi kai lock tou mutex tis add*/
58
59      while(condition_check == 0){                /*elegxos an to condition check einai == 1 , oso einai ==0 tote to ekastote nima kanei wait   */
60          pthread_cond_wait(&cond ,&put_locker);  /*menei edw i sinartisi mexri kapoio allo nima na kanei signal()*/
61      }
62      condition_check = 0;                        /*vazoume to condition_check=0 wste to epomeno nima na min dei ==1 kai perasei tin while*/
63      if (memtable_needs_compaction(self->memtable))
64      {
65          INFO("Starting compaction of the memtable after %d insertions and %d deletions",
66               self->memtable->add_count, self->memtable->del_count);
67          sst_merge(self->sst, self->memtable);
68          memtable_reset(self->memtable);
69      }
70      returnVariable = memtable_add(self->memtable, key, value);      /*arxika to memtable_add(self->memtable, key, value) itan stin return katw alla to valame stin krisimi perioxi*/
71      condition_check = 1;                                             /*thetoume to condition_check=1 wste otan to dei to epomeno nima na figei apo tin while*/
72      pthread_cond_signal(&cond);                                      /*kanei signal ena allo tixaio nima na ksanacheckarei tin sinthiki tis while */
73      pthread_mutex_unlock(&put_locker);                               /*unlock to mutex tis add*/
74
75
76      return returnVariable;
77  }
```

Μέσα στην db_add() δηλώνουμε μια τοπική μεταβλητή , η οποία στην συνέχεια θα αποθηκεύσει την τιμή που θα επιστρέψει η memtable_add() και αυτό μας επιτρέπει να την βάλουμε μέσα σε mutex. Μετά κάνουμε initialization και lock το mutex . Στην “while{}” ελέγχουμε αν το condiotion_check == 0 που σημαίνει ότι κάποιο άλλο νήμα προσθέτει το ζεύγος κλειδί-τιμή στο database, όπου σε αυτήν την περίπτωση αναγκάζουμε το νήμα αυτό να περιμένει (pthread_cond_wait()). Μόλις τελειώσει η “if{}” στέλνουμε την τιμή που επιστρέφει η memtable_add() στην μεταβλητή returnVariable, βάζουμε το condition check=1(ώστε να μπορέσει να βγει από την “while{}” ένα άλλο νήμα , όταν γίνει η κλήση της pthread_signal())καλούμε την pthread_signal() και κάνουμε unlock το mutex ώστε να μπορέσει να το λοκάρει το νήμα που “ξύπνησε από την signal”.

```
79  int db_get(DB* self, Variant* key, Variant* value)
80  {
81      int returnVariable;                                 /*idia xrisi tis topikis metavlitis return variable me tin db_add*/
82
83      pthread_mutex_init(&get_locker, NULL);              /* initialize kai lock tou mutex get_locker*/
84      pthread_mutex_lock(&get_locker);
85      if (memtable_get(self->memtable->list, key, value) == 1)
86          return 1;
87
88      returnVariable = sst_get(self->sst, key, value);    /*apothikefsi tis timis epistrofis tis sst_get sto return variable*/
89      pthread_mutex_unlock(&get_locker);                  /*unlock to mutex*/
90
91      return returnVariable;
92  }
```

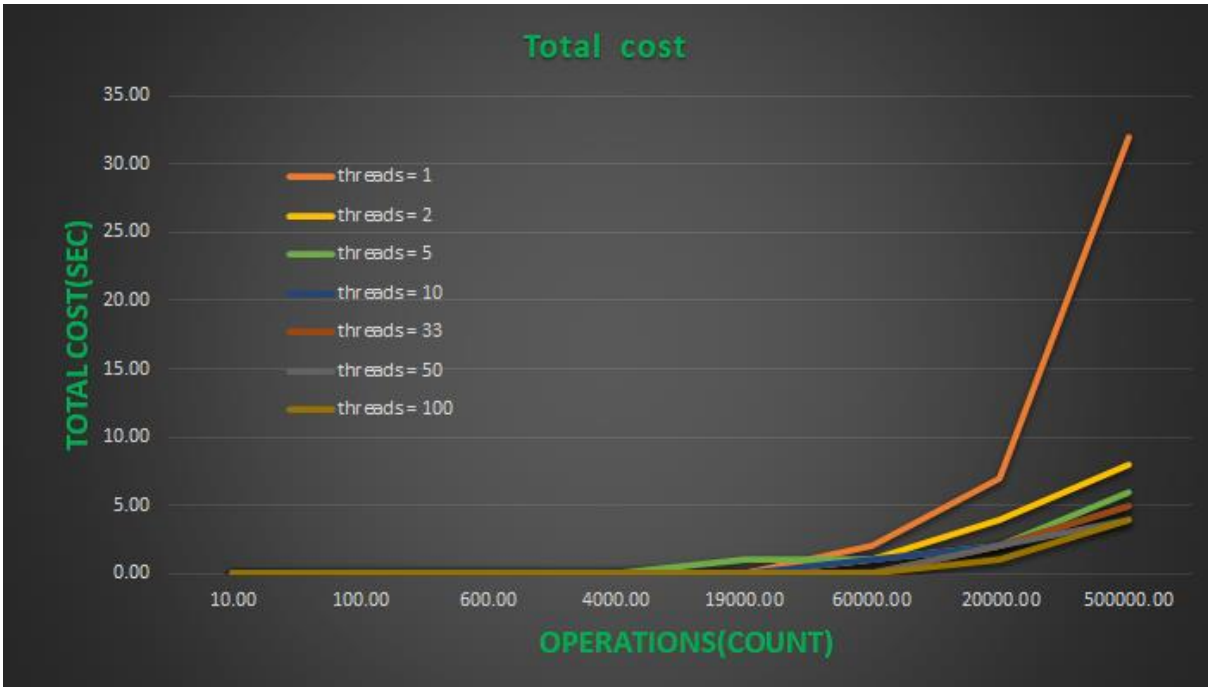
Όπως και στην db_add() έτσι και εδώ δηλώνουμε μια τοπική μεταβλητή returnVariable , κάνουμε initialization και lock το mutex (get_locker) . Μετά Μόλις τελειώσει η “if{}” στέλνουμε την τιμή που επιστρέφει η sst_get() στην μεταβλητή returnVariable και ξεκλειδώνουμε το mutex.

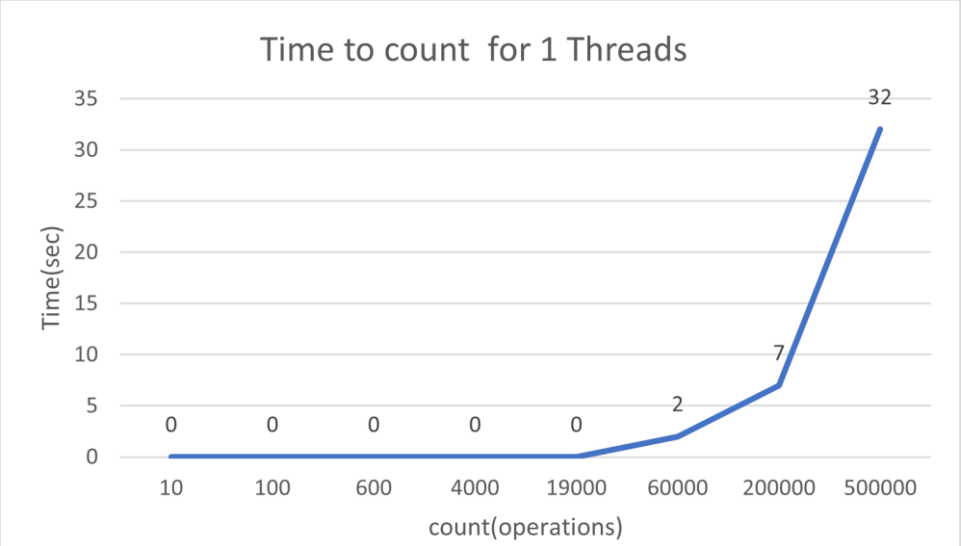
Με αυτό τον τρόπο εξασφαλίζουμε ότι λειτουργούν πολλαπλοί αναγνώστες(db_get()) η ένας γραφέας(db_add()) κάθε φορά .

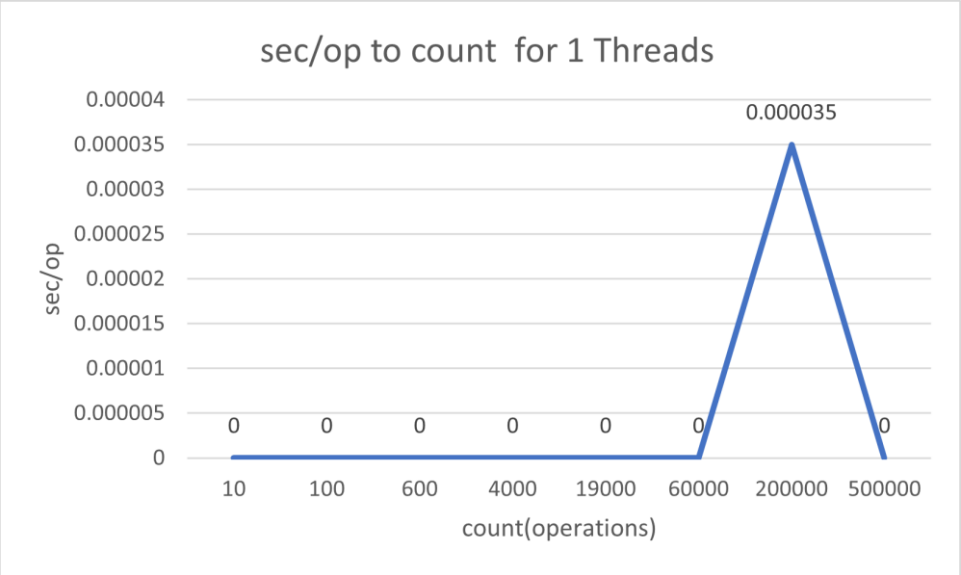
Στατιστικά:

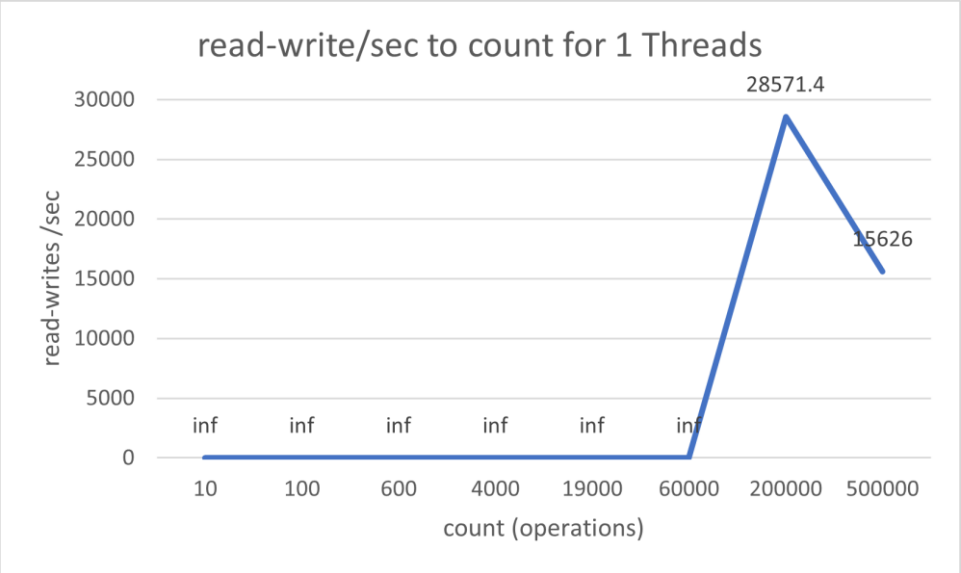
Δοκιμάσαμε το πρόγραμμα για 7 διαφορετικές τιμές threads και 8 τιμές count και μετρήσαμε τις τιμές total cost, sec/op και reads-writes/sec(estimated) και αυτά είναι τα γραφήματα που προέκυψαν από κάτω είναι ένα συνολικό διάγραμμα και ακολουθούν τα επιμέρους για κάθε πλήθος threads:

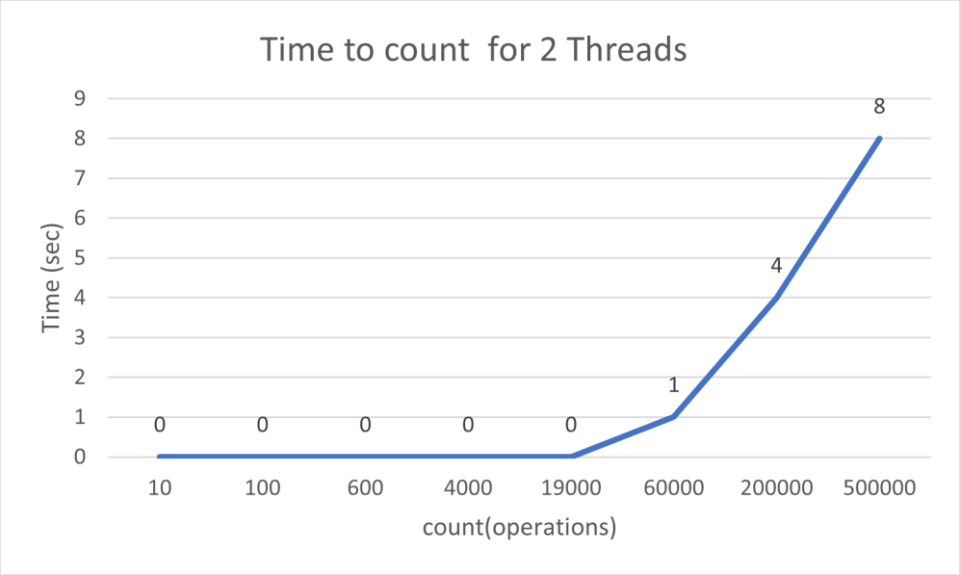
Συγκεντρωτικό δίαγραμμα για το συνολικό κόστος

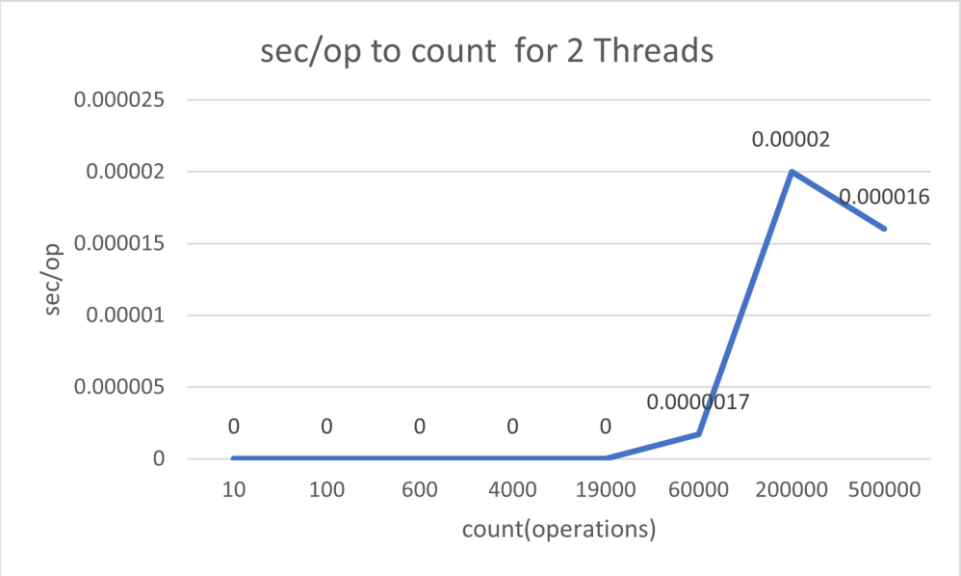


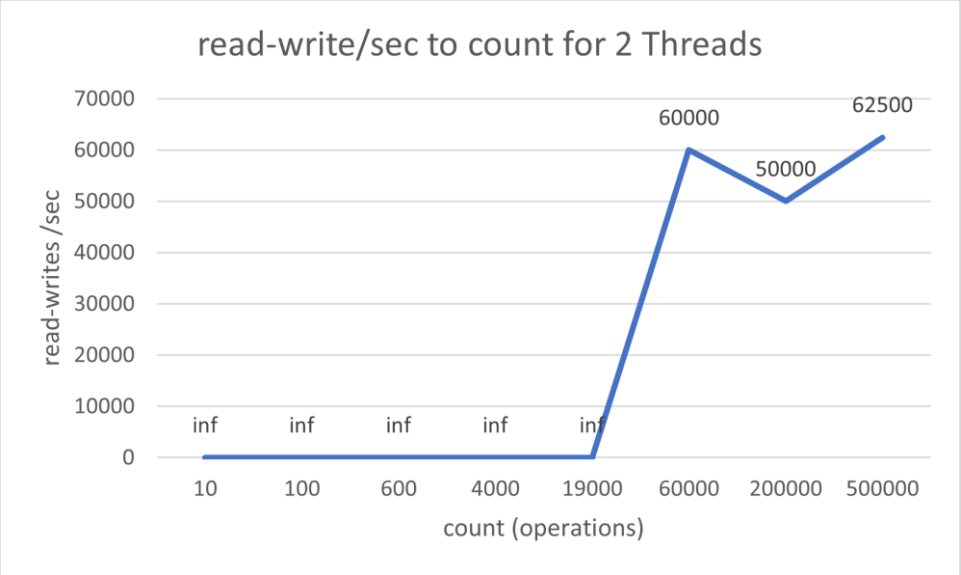

Time to count for 1 Threads
Time(sec)
35
30
25
20
15
10
5
0
0
0
0
0
0
2
7
32
10
100
600
4000
19000
60000
200000
500000
count(operations)

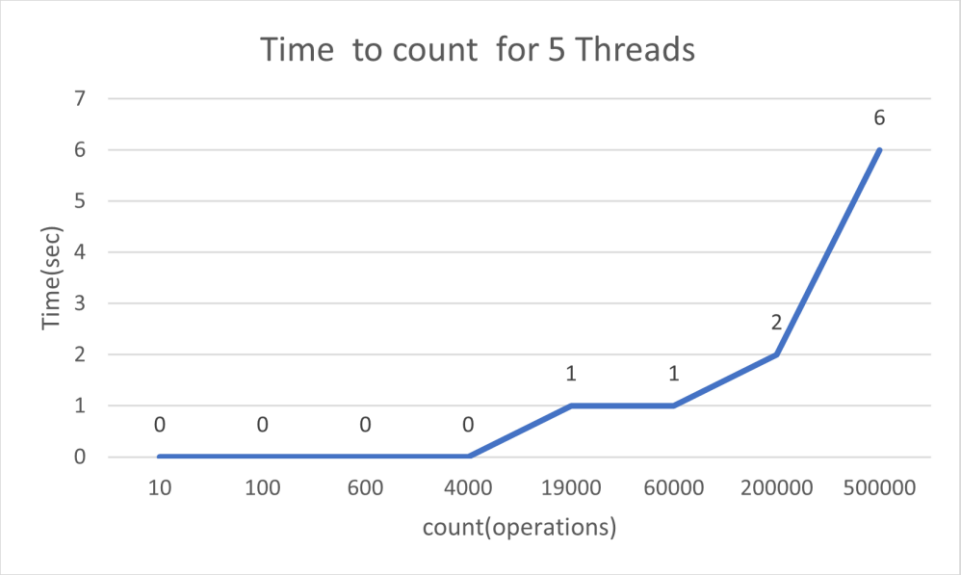

sec/op to count for 1 Threads
sec/op
0.00004
0.000035
0.00003
0.000025
0.00002
0.000015
0.00001
0.000005
0
0
0
0
0
0
0.000035
0
10
100
600
4000
19000
60000
200000
500000
count(operations)

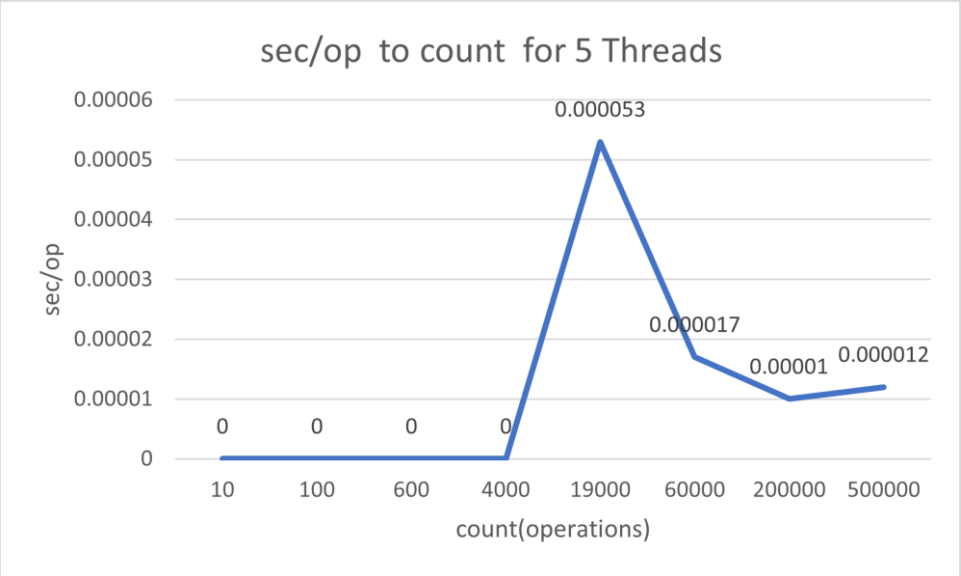

read-write/sec to count for 1 Threads
read-writes /sec
30000
25000
20000
15000
10000
5000
0
inf
inf
inf
inf
inf
28571.4
15626
10
100
600
4000
19000
60000
200000
500000
count (operations)

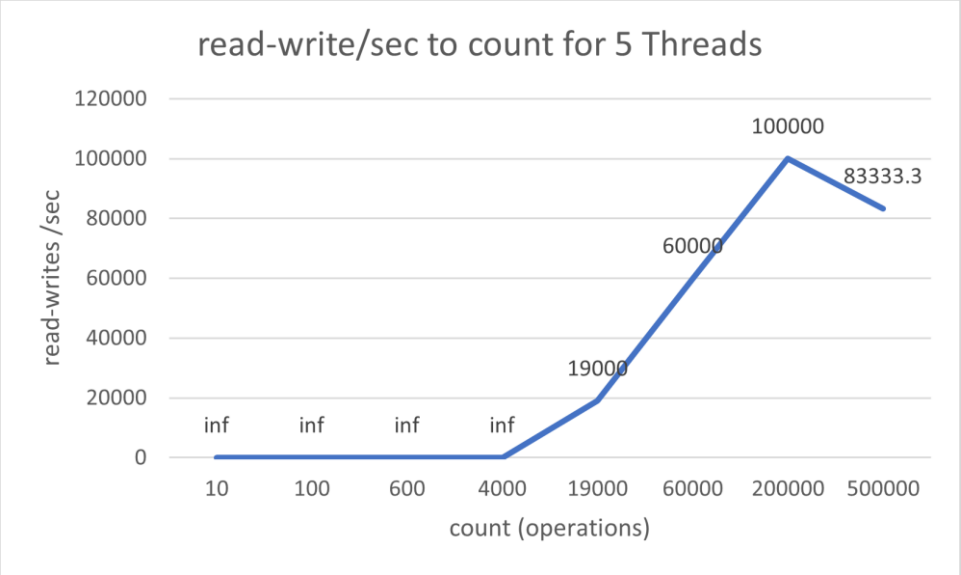

Time to count for 2 Threads
Time (sec)
9
8
7
6
5
4
3
2
1
0
0
0
0
0
0
1
4
8
10
100
600
4000
19000
60000
200000
500000
count(operations)

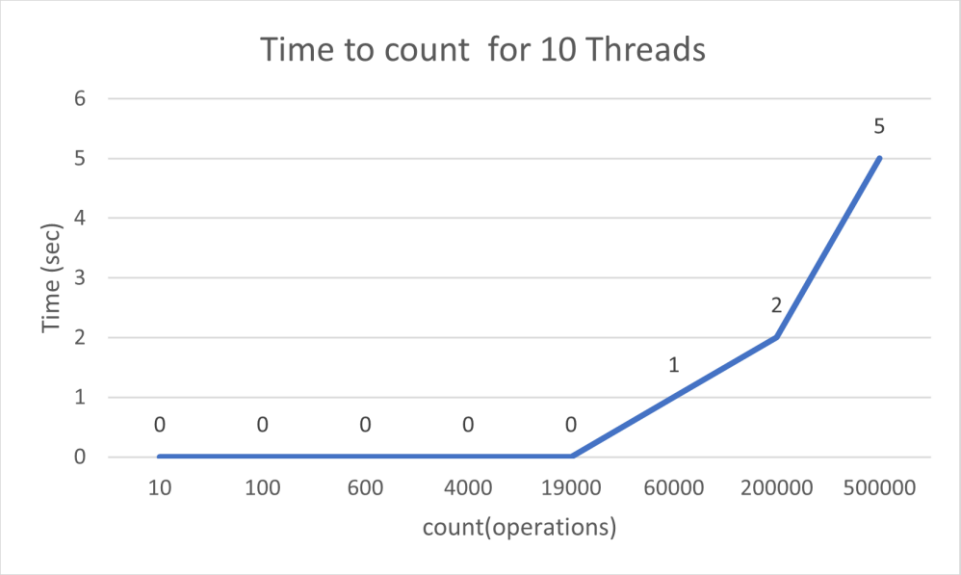

sec/op to count for 2 Threads
sec/op
0.000025
0.00002
0.000015
0.00001
0.000005
0
0
0
0
0
0
0.0000017
0.00002
0.000016
10
100
600
4000
19000
60000
200000
500000
count(operations)

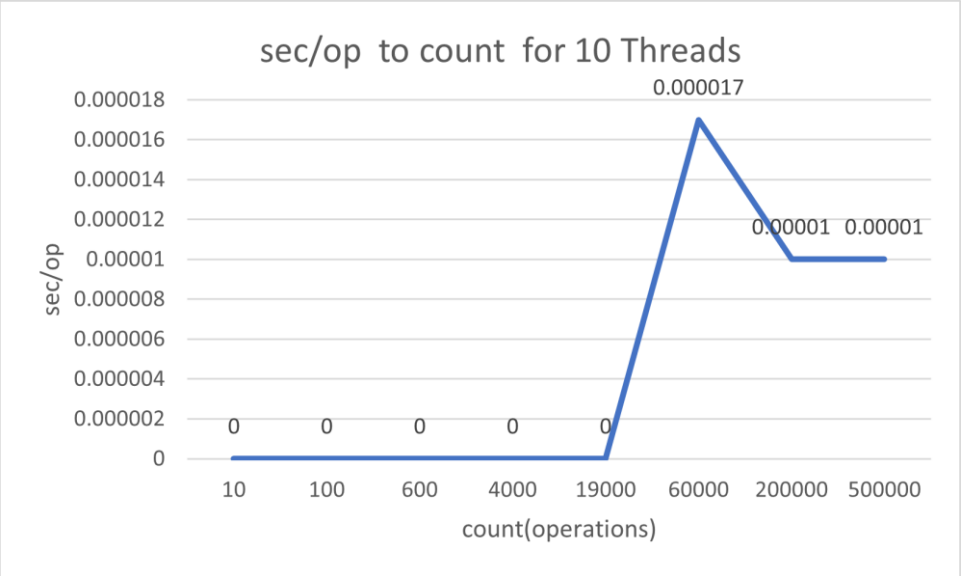

read-write/sec to count for 2 Threads
read-writes /sec
70000
60000
50000
40000
30000
20000
10000
0
inf
inf
inf
inf
inf
60000
50000
62500
10
100
600
4000
19000
60000
200000
500000
count (operations)

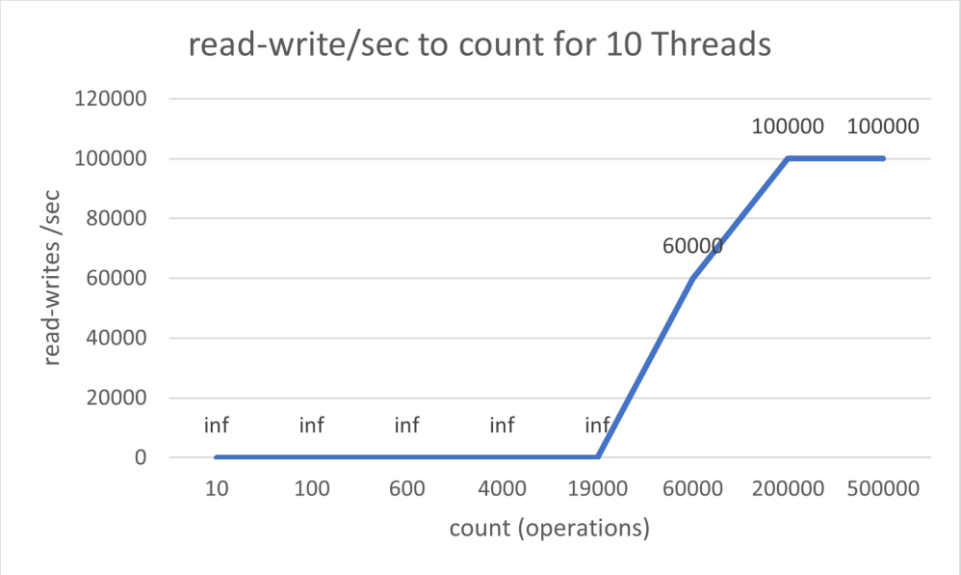

Time to count for 5 Threads
Time(sec)
7
6
5
4
3
2
1
0
0
0
0
0
1
1
2
6
10
100
600
4000
19000
60000
200000
500000
count(operations)

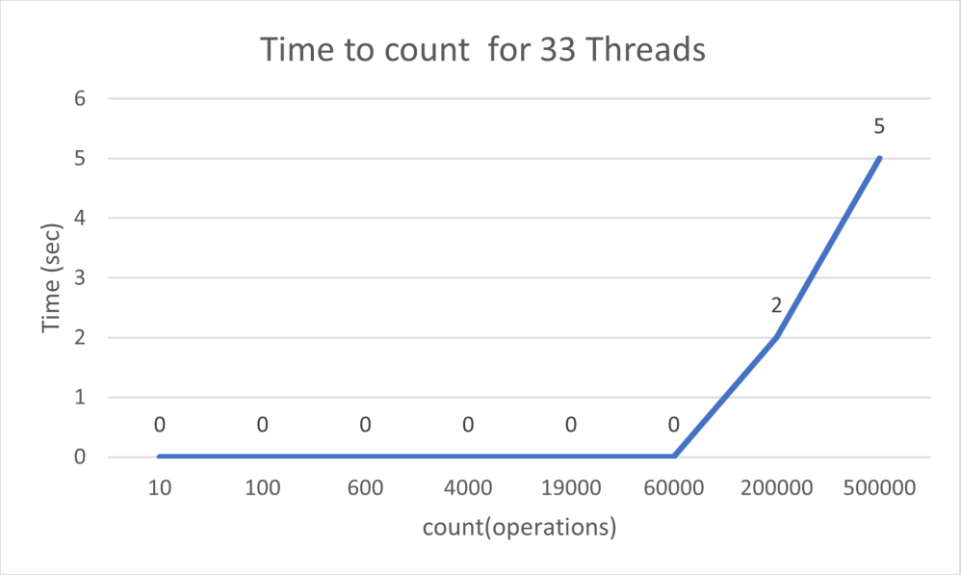

sec/op to count for 5 Threads
sec/op
0.00006
0.00005
0.00004
0.00003
0.00002
0.00001
0
0
0
0
0
0.000053
0.000017
0.00001
0.000012
10
100
600
4000
19000
60000
200000
500000
count(operations)

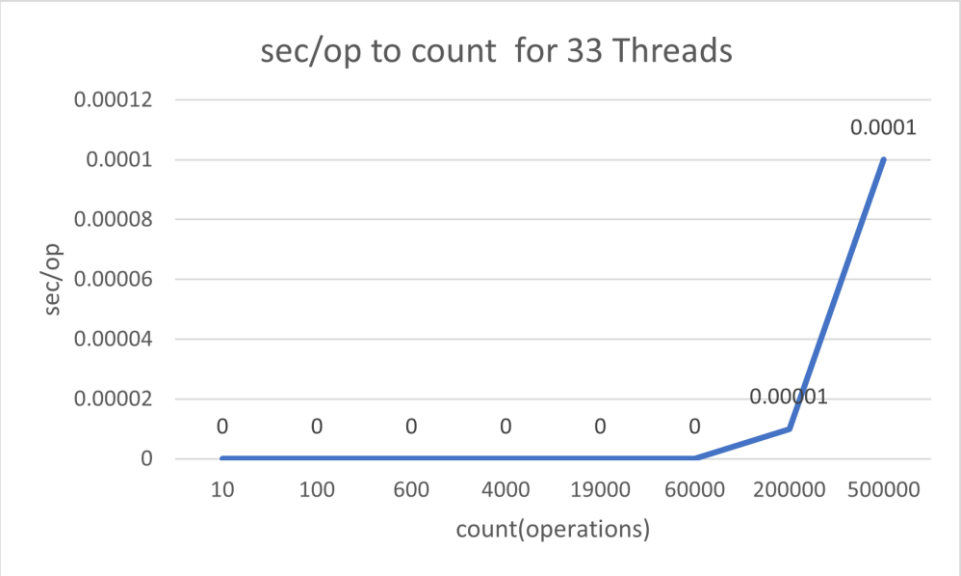

read-write/sec to count for 5 Threads
read-writes /sec
120000
100000
80000
60000
40000
20000
0
inf
inf
inf
inf
19000
60000
100000
83333.3
10
100
600
4000
19000
60000
200000
500000
count (operations)

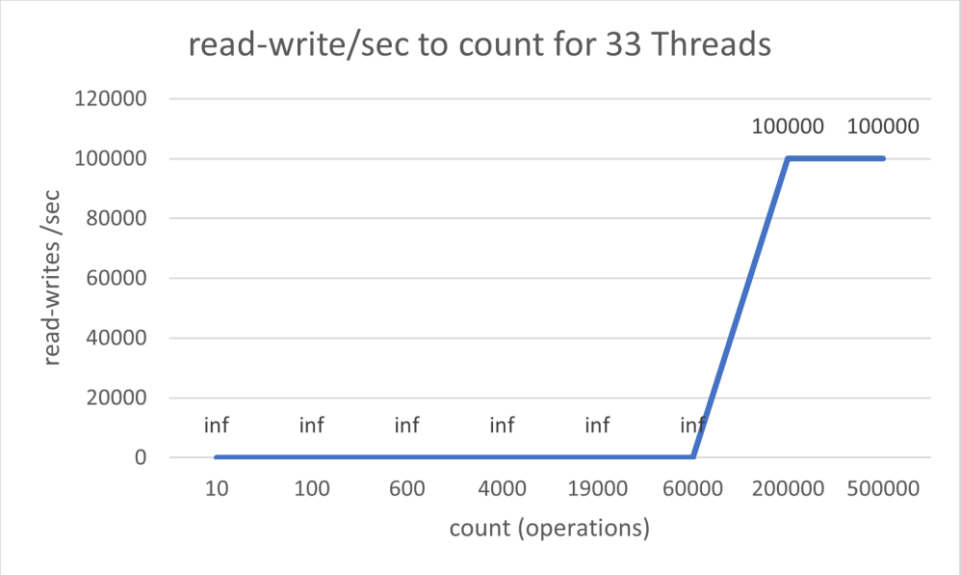

Time to count for 10 Threads
Time (sec)
6
5
4
3
2
1
0
0
0
0
0
0
1
2
5
10
100
600
4000
19000
60000
200000
500000
count(operations)

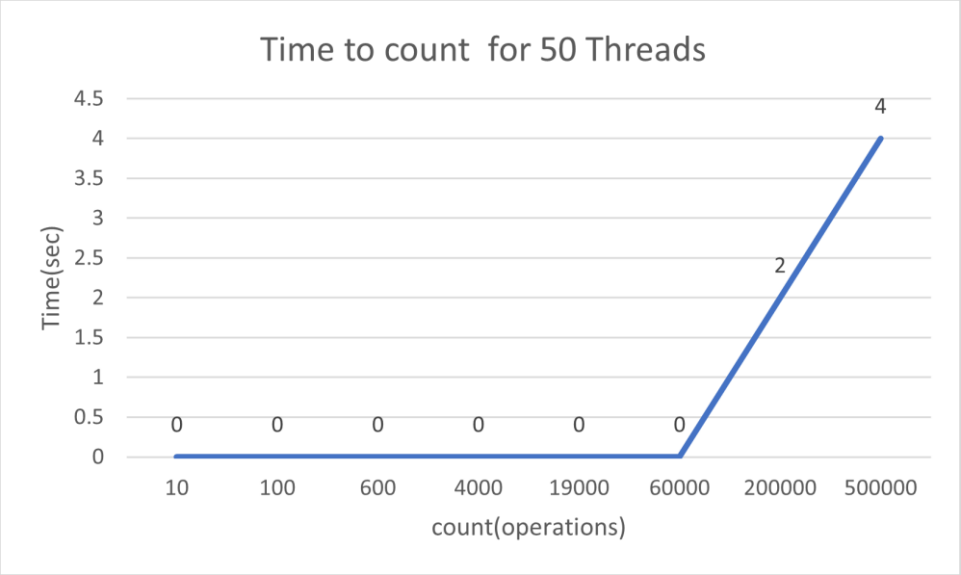

sec/op to count for 10 Threads
sec/op
0.000018
0.000016
0.000014
0.000012
0.00001
0.000008
0.000006
0.000004
0.000002
0
0
0
0
0
0
0.000017
0.00001
0.00001
10
100
600
4000
19000
60000
200000
500000
count(operations)

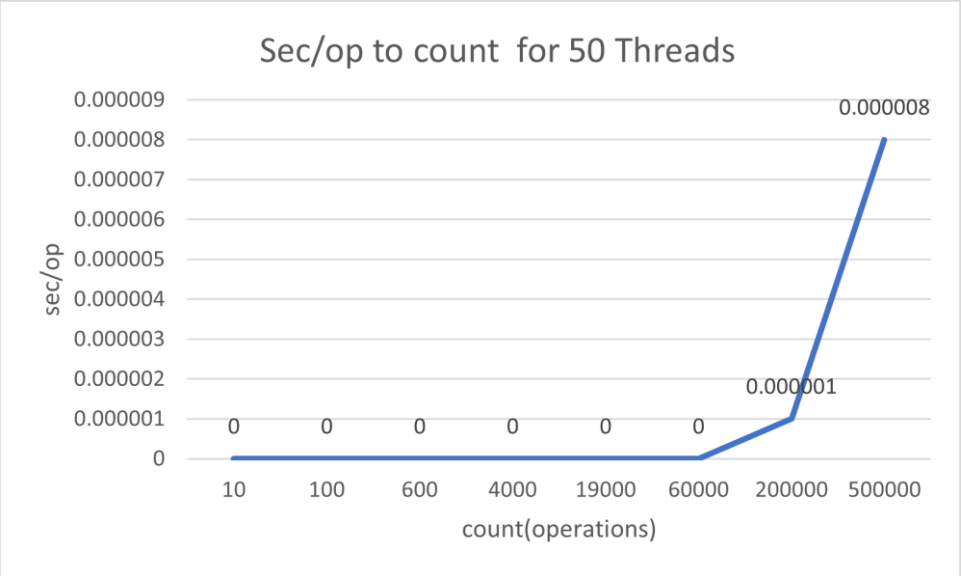

read-write/sec to count for 10 Threads
read-writes /sec
120000
100000
80000
60000
40000
20000
0
inf
inf
inf
inf
inf
60000
100000
100000
10
100
600
4000
19000
60000
200000
500000
count (operations)

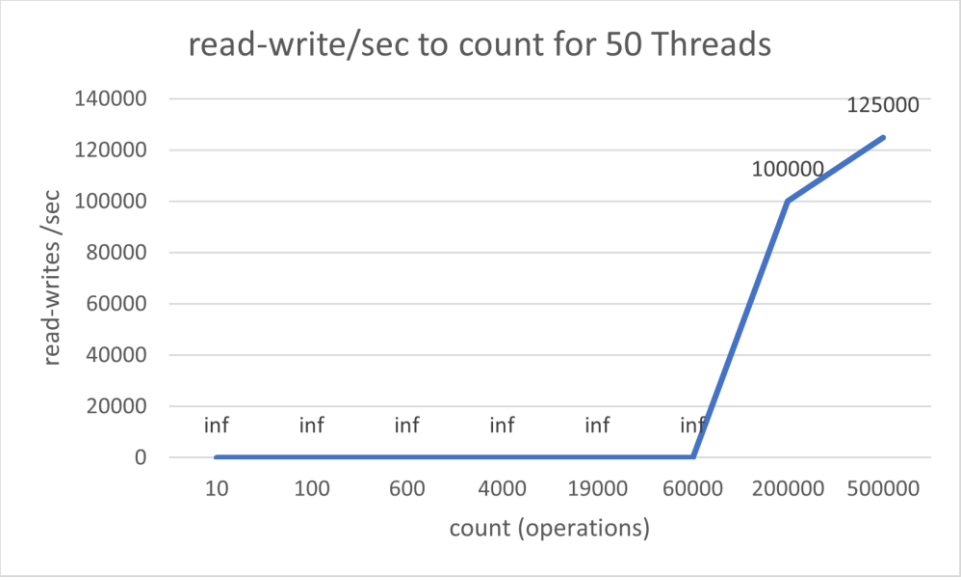

Time to count for 33 Threads
Time (sec)
6
5
4
3
2
1
0
0
0
0
0
0
0
2
5
10
100
600
4000
19000
60000
200000
500000
count(operations)

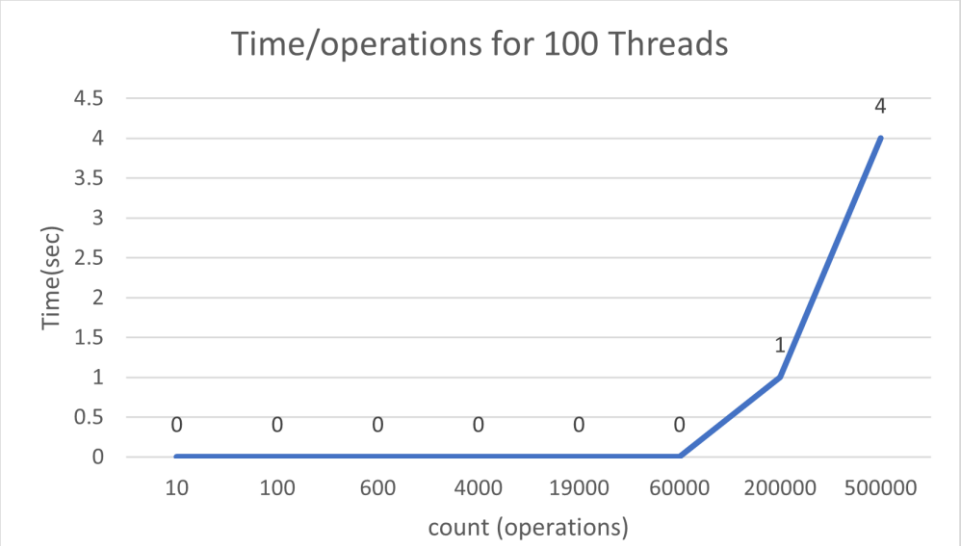

sec/op to count for 33 Threads
sec/op
0.00012
0.0001
0.00008
0.00006
0.00004
0.00002
0
0
0
0
0
0
0
0.00001
0.0001
10
100
600
4000
19000
60000
200000
500000
count(operations)

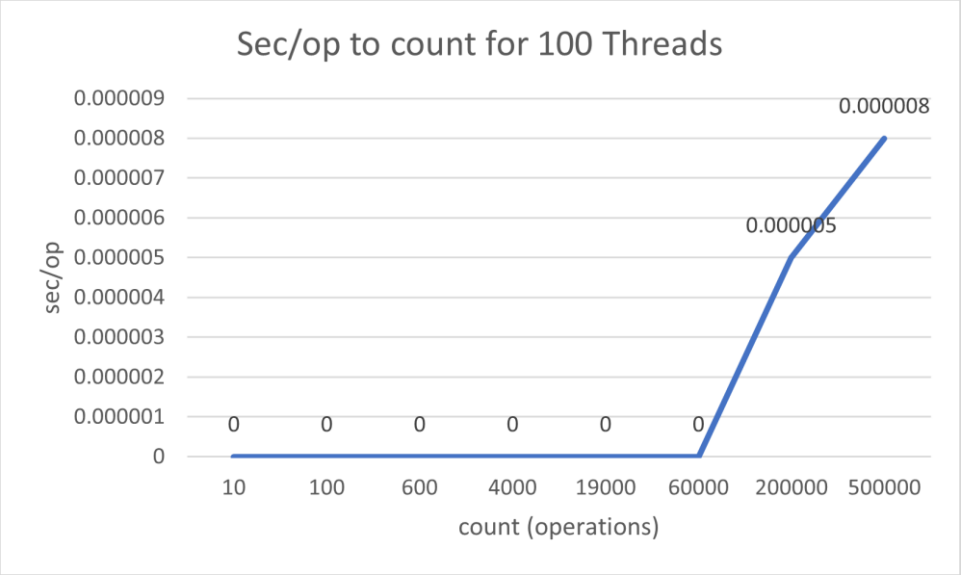

read-write/sec to count for 33 Threads
read-writes /sec
120000
100000
80000
60000
40000
20000
0
inf
inf
inf
inf
inf
inf
100000
100000
10
100
600
4000
19000
60000
200000
500000
count (operations)

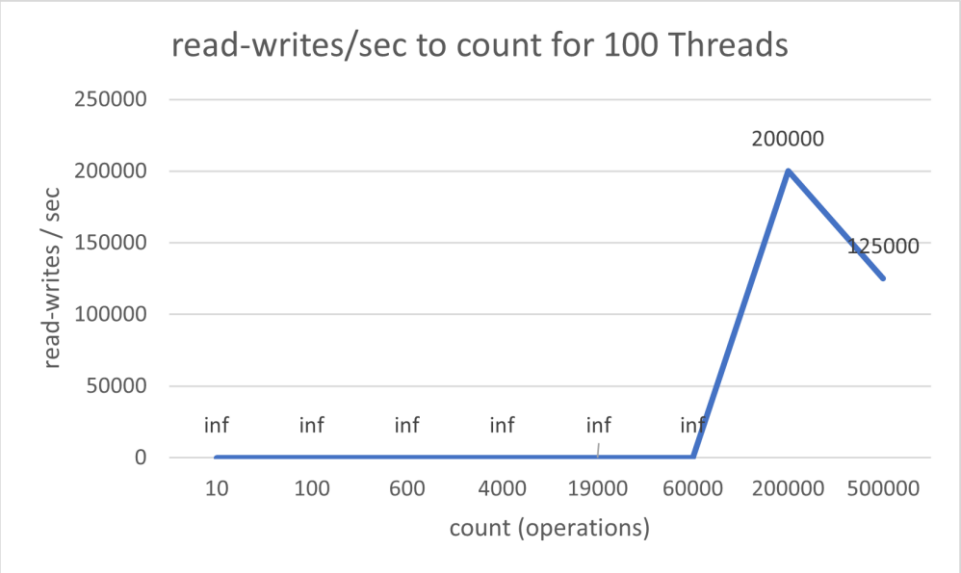

Time to count for 50 Threads
Time(sec)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0
0
0
0
0
2
4
10
100
600
4000
19000
60000
200000
500000
count(operations)


Sec/op to count for 50 Threads
sec/op
0.000009
0.000008
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0
0
0
0
0
0
0
0.000001
0.000008
10
100
600
4000
19000
60000
200000
500000
count(operations)


read-write/sec to count for 50 Threads
read-writes /sec
140000
120000
100000
80000
60000
40000
20000
0
inf
inf
inf
inf
inf
inf
100000
125000
10
100
600
4000
19000
60000
200000
500000
count (operations)


Time/operations for 100 Threads
Time(sec)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0
0
0
0
0
1
4
10
100
600
4000
19000
60000
200000
500000
count (operations)


Sec/op to count for 100 Threads
sec/op
0.000009
0.000008
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0
0
0
0
0
0
0
0.000005
0.000008
10
100
600
4000
19000
60000
200000
500000
count (operations)


read-writes/sec to count for 100 Threads
read-writes / sec
250000
200000
150000
100000
50000
0
inf
inf
inf
inf
inf
inf
200000
125000
10
100
600
4000
19000
60000
200000
500000
count (operations)

Στιγμιότυπα εκτέλεσης προγράμματος:

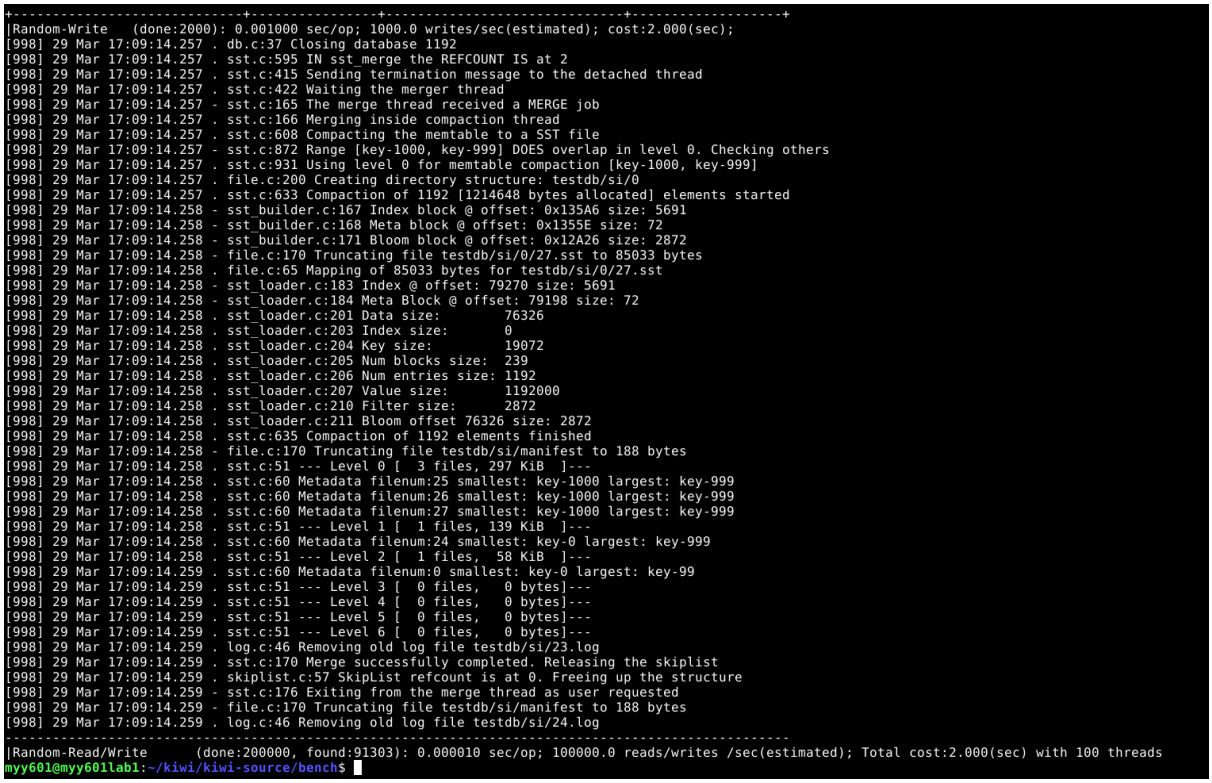
Εκτέλεση make all :

```
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 -lpthread -lsnappy -o kiwi-bench
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$
```

Για είσοδο: ./kiwi-bench readwrite 60000 50

```
+--------------------------+--------------+---------------------------+-----------------+
|Random-Write   (done:1200): 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
[833] 29 Mar 17:07:04.928 . db.c:37 Closing database 1181
[833] 29 Mar 17:07:04.928 . sst.c:595 IN sst_merge the REFCOUNT IS at 2
[833] 29 Mar 17:07:04.928 . sst.c:415 Sending termination message to the detached thread
[833] 29 Mar 17:07:04.928 . sst.c:422 Waiting the merger thread
[833] 29 Mar 17:07:04.928 - sst.c:165 The merge thread received a MERGE job
[833] 29 Mar 17:07:04.928 . sst.c:166 Merging inside compaction thread
[833] 29 Mar 17:07:04.928 . sst.c:608 Compacting the memtable to a SST file
[833] 29 Mar 17:07:04.928 - sst.c:872 Range [key-1000, key-999] DOES overlap in level 0. Checking others
[833] 29 Mar 17:07:04.928 . sst.c:931 Using level 0 for memtable compaction [key-1000, key-999]
[833] 29 Mar 17:07:04.928 . file.c:200 Creating directory structure: testdb/si/0
[833] 29 Mar 17:07:04.928 . sst.c:633 Compaction of 519 [528861 bytes allocated] elements started
[833] 29 Mar 17:07:04.928 - sst_builder.c:167 Index block @ offset: 0x8735 size: 2451
[833] 29 Mar 17:07:04.928 - sst_builder.c:168 Meta block @ offset: 0x86ED size: 72
[833] 29 Mar 17:07:04.928 - sst_builder.c:171 Bloom block @ offset: 0x8209 size: 1252
[833] 29 Mar 17:07:04.928 - file.c:170 Truncating file testdb/si/0/46.sst to 37136 bytes
[833] 29 Mar 17:07:04.929 . file.c:65 Mapping of 37136 bytes for testdb/si/0/46.sst
[833] 29 Mar 17:07:04.929 - sst_loader.c:183 Index @ offset: 34613 size: 2451
[833] 29 Mar 17:07:04.929 - sst_loader.c:184 Meta Block @ offset: 34541 size: 72
[833] 29 Mar 17:07:04.929 . sst_loader.c:201 Data size:        33289
[833] 29 Mar 17:07:04.929 . sst_loader.c:203 Index size:       0
[833] 29 Mar 17:07:04.929 . sst_loader.c:204 Key size:         8304
[833] 29 Mar 17:07:04.929 . sst_loader.c:205 Num blocks size:  104
[833] 29 Mar 17:07:04.929 . sst_loader.c:206 Num entries size: 519
[833] 29 Mar 17:07:04.929 . sst_loader.c:207 Value size:       519000
[833] 29 Mar 17:07:04.929 . sst_loader.c:210 Filter size:      1252
[833] 29 Mar 17:07:04.929 . sst_loader.c:211 Bloom offset 33289 size: 1252
[833] 29 Mar 17:07:04.929 . sst.c:635 Compaction of 519 elements finished
[833] 29 Mar 17:07:04.929 - file.c:170 Truncating file testdb/si/manifest to 260 bytes
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 0 [  5 files, 357 KiB  ]---
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:42 smallest: key-10 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:43 smallest: key-10 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:44 smallest: key-100 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:45 smallest: key-1000 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:46 smallest: key-1000 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 1 [  1 files,   2 MiB  ]---
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:41 smallest: key-0 largest: key-9999
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 2 [  1 files, 133 KiB  ]---
[833] 29 Mar 17:07:04.929 . sst.c:60 Metadata filenum:0 smallest: key-0 largest: key-999
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 3 [  0 files,   0 bytes]---
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 4 [  0 files,   0 bytes]---
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 5 [  0 files,   0 bytes]---
[833] 29 Mar 17:07:04.929 . sst.c:51 --- Level 6 [  0 files,   0 bytes]---
[833] 29 Mar 17:07:04.929 . log.c:46 Removing old log file testdb/si/7.log
[833] 29 Mar 17:07:04.929 . sst.c:170 Merge successfully completed. Releasing the skiplist
[833] 29 Mar 17:07:04.929 . skiplist.c:57 SkipList refcount is at 0. Freeing up the structure
[833] 29 Mar 17:07:04.929 - sst.c:176 Exiting from the merge thread as user requested
[833] 29 Mar 17:07:04.930 - file.c:170 Truncating file testdb/si/manifest to 260 bytes
[833] 29 Mar 17:07:04.930 . log.c:46 Removing old log file testdb/si/8.log
-------------------------------------------------------------------------------------------
|Random-Read/Write      (done:60000, found:30000): 0.000000 sec/op; inf reads/writes /sec(estimated); Total cost:0.000(sec) with 50 threads
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Για είσοδο: ./kiwi-bench readwrite 200000 100



Σχόλια-Παρατηρήσεις:

- Καταφέραμε να κάνουμε την πρώτη υλοποίηση (με το το καθολικό lock για τα db_get και db_add) και έπειτα το εξελίξαμε στην τρέχουσα μορφή του.
- Στο bench.c βάλαμε μια επιπλέον μεταβλητή με όνομα ‘percentage’ η οποία ωστόσο δεν αξιοποιείται πουθενά στον κώδικα καθώς προοριζόταν για να δίνει ο χρήστης μια 5^{η} παράμετρο στην ./kiwi-bench η οποία θα καθόριζε το ποσοστό των λειτουργιών από κάθε τύπο (put/get)
- Καταφέραμε να ελαχιστοποιήσουμε το πρόβλημα που είχαμε με την διαίρεση άρτιων με περιττών(και αντίστροφα) όπου χάνονταν κάποια operations ωστόσο ακόμα και τώρα χάνεται το πολύ 1 operation.