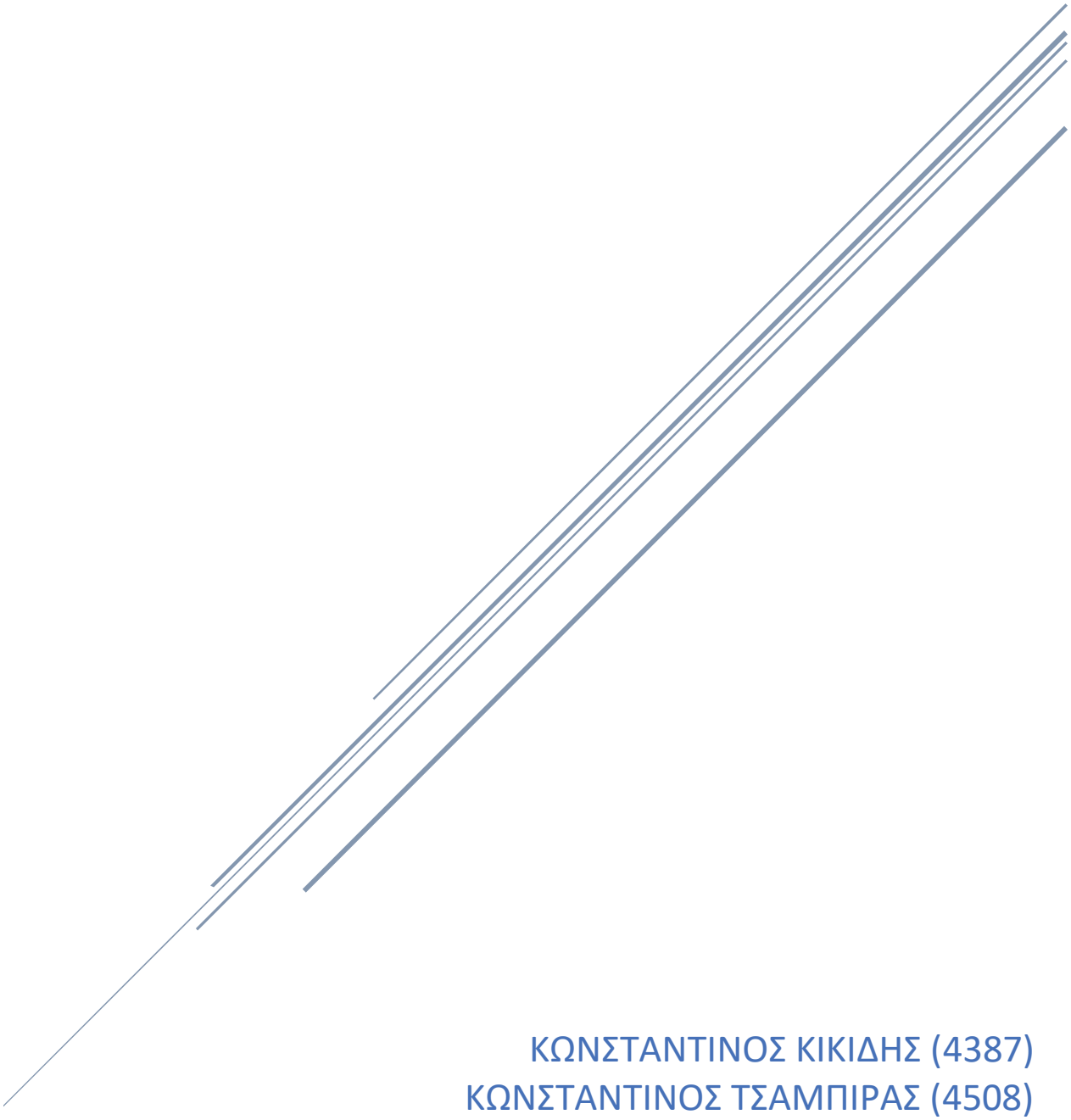
ΑΝΑΦΟΡΑ

Εργαστήριο 1: Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων



ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΙΚΙΔΗΣ (4387)
ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΣΑΜΠΙΡΑΣ (4508)

Ιστορικό Εκδόσεων

Έκδοση	Ημερ/νία	Αλλαγές
1.0	8/3/2021	• Αρχικός κώδικας.
1.1	14/3/2021	• Δημιουργία νέου νήματος για την _write_test.
1.2	14/3/2021	• Δημιουργία πολυνηματικής υλοποίησης για την _read_test.
1.2.1	15/3/2021	• Μοίρασμα της ανάγνωσης σε κάθε νήμα, πχ για 100 read με 4 νήματα, κάθε νήμα θα έχει από 25 read (0-24, 25-49, 50-74, 75-99).
1.2.2	16/3/2021	• Δημιουργία νέου αρχείου για καταχώριση στατιστικών απόδοσης (read και write).
1.2.3	18/3/2021	• Επέκταση της υλοποίησης σε περίπτωση που το count δεν διαιρείται ακριβώς με τον αριθμό των threads, πχ για 101, 102, 103 read και 4 νήματα, κάθε νήμα θα αναλάβει από 25 read, εκτός από το τελευταίο που θα κάνει 26, 27, 28 αντίστοιχα. Το τελευταίο thread θα αναλάβει να διαβάσει ό,τι έχει μείνει. • Προσθήκη σχολίων στον κώδικα.
1.2.4	22/3/2021	• Πρόσθεση μερικών νέων mutex_lock σε κάποιες περιοχές του κώδικα. Βελτίωση αξιοπιστίας _read_test • Χρήση ενός νέου μετρητή, του threadCounter. • Αλλαγή μερικών μεταβλητών σε global και προσθήκη νέων σχολίων.
1.2.5	24/3/2021	• Διόρθωση σφαλμάτων στην _read_test, πλέον δουλεύει σωστά.
1.3	24/3/2021	• Δημιουργία πολυνηματικής υλοποίησης για την _write_test.
1.3.1	25/3/2021	• Βελτίωση αξιοπιστίας της _write_test (χάνει λίγα κλειδιά, 2-3 στα 100000).
1.3.2	27/3/2021	• Διόρθωση σφαλμάτων στην _write_test, πλέον δουλεύει σωστά.
1.4	28/3/2021	• Δημιουργία μίας νέας επιλογής, της readwrite, η οποία γράφει και διαβάζει ταυτόχρονα. Υπάρχουν προβλήματα (Segmentation Fault). • Προσθήκη σχολίων στον κώδικα. • Χρήση ενός νέου μετρητή, του writeFlag.
1.4.1	28/3/2021	• Ολοκλήρωση της readwrite, δουλεύει σωστά.
1.5	29/3/2021	• Τελική έκδοση. • Αφαίρεση κώδικα αποσφαλμάτωσης. • Για κάθε νήμα, τυπώνουμε σε αρχείο, τα ατομικά στατιστικά απόδοσης. • Προσθήκη σχολίων.

Αρχεία που άλλαξαν

- kiwi.c
- bench.h
- bench.c

Αλλαγές που πραγματοποιήθηκαν

- Πολυνηματική υλοποίηση της read
- Πολυνηματική υλοποίηση της write
- Ταυτόχρονη κλήση των read και write με την readwrite

Προβλήματα που υπήρχαν

Πολυνηματική read

Κατά την πολυνηματική υλοποίηση της _read_test (στις εκδόσεις 1.2.1 - 1.2.3) υπάρχουν ορισμένες φορές προβλήματα κατά την εκτέλεση. Συγκεκριμένα, κάποιες φορές, το πρόγραμμα μας σταμάταγε την εκτέλεση με το σφάλμα Bus Error, αυτό όμως δεν συνέβαινε σε κάθε εκτέλεση. Σε περίπτωση επιτυχούς εκτέλεσης είχαμε αυτά τα αποτελέσματα (για αρκετά μεγάλους αριθμούς, υπάρχει αισθητή διαφορά στον χρόνο εκτέλεσης της μονονηματικής write και της πολυνηματικής (4 thread) read).

```
|Random-Write   (done:1000000): 0.000012 sec/op; 83333.3 writes/sec(estimated); cost:12.000(sec);
|Random-Read    (done:1000000, found:250000): 0.000008 sec/op; 125000.0 reads /sec(estimated); cost:8.000(sec)
|Random-Read    (done:1000000, found:250000): 0.000008 sec/op; 125000.0 reads /sec(estimated); cost:8.000(sec)
|Random-Read    (done:1000000, found:250000): 0.000008 sec/op; 125000.0 reads /sec(estimated); cost:8.000(sec)
|Random-Read    (done:1000000, found:250000): 0.000006 sec/op; 166666.7 reads /sec(estimated); cost:6.000(sec)

|Random-Write   (done:2000000): 0.000017 sec/op; 58823.5 writes/sec(estimated); cost:34.000(sec);
|Random-Read    (done:2000000, found:500000): 0.000008 sec/op; 125000.0 reads /sec(estimated); cost:16.000(sec)
|Random-Read    (done:2000000, found:500000): 0.000008 sec/op; 125000.0 reads /sec(estimated); cost:16.000(sec)
|Random-Read    (done:2000000, found:500000): 0.000008 sec/op; 133333.3 reads /sec(estimated); cost:15.000(sec)
|Random-Read    (done:2000000, found:500000): 0.000006 sec/op; 153846.2 reads /sec(estimated); cost:13.000(sec)
```

Αποτελέσματα επιτυχούς εκτέλεσης

Αλλά πολλές φορές το αποτέλεσμα της _read_test ήταν και αυτό…

```
[1047] 21 Mar 11:48:54.546 . sst_loader.c:206 Num entries size: 4117
[1047] 21 Mar 11:48:54.546 . sst_loader.c:207 Value size:       4117000
[1047] 21 Mar 11:48:54.546 . sst_loader.c:210 Filter size:      9892
[1047] 21 Mar 11:48:54.546 . sst_loader.c:211 Bloom offset 266433 size: 9892
[1047] 21 Mar 11:48:54.547 . sst.c:342 Smallest key: key-0 Largest key: key-999999 seeks: 100
Bus error
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Κατά την κανονική εκτέλεση από το τερματικό

```
26 searching key-26
[1110] 21 Mar 11:52:07.268 . sst.c:60 Metadata filenum:269 smallest: key-934559 largest: key-938675
[1110] 21 Mar 11:52:07.268 . sst.c:60 Metadata filenum:270 smallest: key-938676 largest: key-942792

Thread 8 "kiwi-bench" received signal SIGBUS, Bus error.
[Switching to Thread 0x7fffe7fff700 (LWP 1117)]
__memmove_avx_unaligned_erms () at ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:468
468     ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S: No such file or directory.
(gdb)
```

Εκτέλεση μέσα από τον GDB

Με αποτέλεσμα, η χρήση της μεθόδου να είναι τουλάχιστον αναξιόπιστη.

Το πρόβλημα αυτό λύθηκε στην έκδοση 1.2.5, κάνοντας global την threadCounter ανοίγοντας και κλείνοντας την βάση μόνο μία φορά, το 1ο νήμα την ανοίγει και το τελευταίο την κλείνει.

Πολυνηματική write

Στην write, είχαμε την ιδέα να σπάμε τον φόρτο σε κάθε νήμα και το κάθε νήμα να βάζει μία τιμή από έναν global counter, αυτό όμως δημιουργούσε θέματα κατά τις εγγραφές καθώς κάποια κλειδιά είτε δεν γράφονταν είτε γράφονταν ξανά (1.3 και 1.3.1).

Η λύση εν τέλει ήρθε όταν κάναμε μία δοκιμή να χρησιμοποιήσουμε την λογική που είχαμε για την read και τελικά δούλεψε σχεδόν αμέσως με μικρές αλλαγές.

Η λειτουργία της πολυνηματικής υλοποίησης για την read

Η βασική ιδέα της υλοποίησης μας είναι πως για όσες καταχωρίσεις θέλουμε να διαβάσουμε (πχ, 100), σπάμε τον φόρτο εργασίας στον αριθμό των threads που δημιουργούμε, πχ για 4 threads και 100 καταχωρίσεις, ο φόρτος εργασίας του κάθε νήματος θα είναι 25, και κάθε νήμα θα έχει να διαβάσει από ένα συγκεκριμένο range [0-25), [25-50), [50-75), [75-100).

Τεκμηρίωση Κώδικα

Για αυτόν τον λόγο, τροποποιήσαμε την read κατάλληλα ώστε να μπορεί να δέχεται πολλαπλά ορίσματα μέσα από ένα struct (bench.h, γραμμές 23-28).

```
struct data {                                   //
        long int count;                         // Ο συνολικός αριθμός καταχωρίσεων
        int r;                                  // Παράμετρος για αναζήτηση άγνωστων κλειδιών
        long int index_start, index_end;        // Το διάστημα που θα διαβάζει το κάθε νήμα
        pthread_t       tid;                    // Το αναγνωριστικό του κάθε νήματος
};                                              //
```

Για τον αριθμό των νημάτων, υπάρχει η numOfThreads (bench.h, γραμμή 16), η οποία έχει γίνει define και μπορεί να αλλάξει. (Επίσης έχει συμπεριληφθεί η βιβλιοθήκη pthreads.h στο bench.h, γραμμή 8).

```
#define numOfThreads 16                         // Αριθμός νημάτων για read και write
```

Η main, όταν κληθεί με το όρισμα read, θα εκτελέσει κάποιες βασικές εντολές και μετά έχουν γίνει κάποιες προσθήκες (bench.c, γραμμές 122-142) για την σωστή λειτουργία της πολυνηματικής ανάγνωσης.

```
struct data thread_args[numOfThreads];                  // Φτιάχνουμε μία λίστα από struct data
long int index_n = count/numOfThreads;                  // Ο φόρτος για το κάθε νήμα
int remainder = count % numOfThreads;                   // Το υπόλοιπο της παραπάνω ακέραιας διαίρεσης
long int index_start = 0;                               // Η αρχική τιμή του 1ου διαστήματος ανάγνωσης
long int index_end = index_start + index_n;             // Η τελική τιμή του 1ου διαστήματος ανάγνωσης
for (int i = 0; i < numOfThreads; i++) {                // Για όσα threads έχουμε
        thread_args[i].count = count;                   // Περνάμε το count σαν παράμετρο
        thread_args[i].r = r;                           // Περνάμε το r σαν παράμετρο
        thread_args[i].index_start = index_start;       // Περνάμε την αρχή του διαστήματος ανάγνωσης σαν παράμετρο
        thread_args[i].index_end = index_end;           // Περνάμε το τέλος του διαστήματος ανάγνωσης σαν παράμετρο
        pthread_create(&thread_args[i].tid, NULL, _read_test, (void *) &thread_args[i]); // Δημιουργούμε το νέο νήμα
        index_start = index_end;                        // Αλλάζουμε τα διαστήματα, τέλος παλιού -> αρχή νέου
        if (i == numOfThreads-2) {                      // Αν το επόμενο είναι το τελευταίο νήμα
                index_end = index_end + index_n + remainder;     // Λάβε υπόψιν το υπόλοιπο που μπορεί να έχει μείνει
        } else {                                        // Αλλιώς
                index_end = index_end + index_n;        // Το τέλος του νέου διαστ., θα είναι το παλιό τέλος +φόρτος
        }                                               //
}                                                       //
for (int i = 0; i < numOfThreads; i++){                 // Μπλοκάρει την εκτέλεση
        pthread_join(thread_args[i].tid, NULL);         // μέχρι να επιστρέψουν όλα τα νήματα
}                                                       //
```

Καλούμε μέσα από τα threads, την _read_test (kiwi.c), εκεί έχουν γίνει οι εξής προσθήκες/αλλαγές.

Δηλώσεις mutexes, conditional variable και global μεταβλητών (kiwi.c, γραμμές 9-22).

```
// Δήλωση των mutex που θα χρησιμοποιηθούν
pthread_mutex_t writeStatsToFile = PTHREAD_MUTEX_INITIALIZER;    // Για την εγγραφή στατιστικών σε αρχείο
pthread_mutex_t dbLock = PTHREAD_MUTEX_INITIALIZER;              // (Για την _write_test) κλείδωμα της εγγραφής.
pthread_mutex_t dbOpenClose = PTHREAD_MUTEX_INITIALIZER;         // Για το άνοιγμα και το κλείσιμο της DB

pthread_mutex_t writeMutex = PTHREAD_MUTEX_INITIALIZER;          // (για την readwrite) κλειδαριά για τους αναγνώστες
pthread_cond_t writeDone = PTHREAD_COND_INITIALIZER;             // (για την readwrite) μεταβλητή συνθήκης


FILE *fp;                       // Δήλωση δείκτη σε αρχείο

DB* db;                         // Δήλωση της βάσης (από τοπική σε global), για κοινή πρόσβαση από όλα τα νήματα
int threadCounter = 0;          // Για το άνοιγμα και το κλείσιμο της βάσης
int writeFlag = 0;              // (readwrite) Μεταβλητή που μετρά τους ενεργούς γραφείς, για να δούμε αν τελείωσαν
```

Περνάμε σε τοπικές μεταβλητές τις παραμέτρους count και r (kiwi.c, γραμμές 129-131).

```
struct data *d = (struct data *) arg;           // Παίρνουμε τις παραμέτρους και τις κάνουμε cast σε struct data
long int count = d->count;                      // Η τοπική count, παίρνει την τιμή count από τις παραμέτρους
int r = d->r;                                   // Αντίστοιχα και η r
```

Ανοίγουμε και κλείνουμε την βάση (μία φορά) (kiwi.c, γραμμές 143-148 και 182-187).

```
pthread_mutex_lock(&dbOpenClose);               // Κλειδώνουμε
threadCounter++;                                // Αυξάνουμε τον μετρητή threadCounter
if (threadCounter == 1) {                       // Αν είναι το 1ο νήμα, τότε
        db = db_open(DATAS);                    // ανοίγει την βάση
}                                               //
pthread_mutex_unlock(&dbOpenClose);             // Ξεκλειδώνουμε


pthread_mutex_lock(&dbOpenClose);               // Κλειδώνουμε
threadCounter--;                                // Μειώνουμε τον μετρητή threadCounter
if (threadCounter == 0) {                       // Αν είναι το τελευταίο νήμα, τότε
        db_close(db);                           // κλείνει την βάση
}                                               //
pthread_mutex_unlock(&dbOpenClose);             // Ξεκλειδώνουμε
```

Το κάθε νήμα εκτελεί μία for loop, με αρχή και τέλος από τα ορίσματα που περάσαμε σε κάθε νήμα (kiwi.c, γρ. 152).

```
// πχ για read 100 με 4 νήματα, κάθε νήμα θα έχει διαφορετικό διάστημα, [0-25), [25-50), [50-75) και [75-100)
for (i = d->index_start; i < d->index_end; i++) {
```

Γράφουμε τα στατιστικά της εκτέλεσης για κάθε νήμα στο αρχείο output.txt (kiwi.c, γραμμές 195-201).

```
// Γράφουμε σε ένα αρχείο, τα στατιστικά της ανάγνωσης
pthread_mutex_lock(&writeStatsToFile);   // Κλειδώνουμε
fp = fopen("output.txt","a+");           // Ανοίγουμε το αρχείο (αν δεν υπάρχει δημιουργείται)
long int threadTotal = d->index_end-d->index_start;      // Υπολογίζουμε το συνολικό φόρτο του κάθε νήματος

// Τυπώνουμε στο αρχείο τα στατιστικά εκτέλεσης του νήματος
fprintf(fp, "Thread %ld completed execution, found %d/%ld reads: %.6f sec/op; %.1f reads/sec(estimated);
cost:%.3f(sec);\n", d->tid, found, threadTotal, (double) (cost/threadTotal), (double) (threadTotal/cost), cost);

fclose(fp);                              // Κλείνουμε το αρχείο
pthread_mutex_unlock(&writeStatsToFile); // Ξεκλειδώνουμε
```

Υπάρχει και ένας έλεγχος στην αρχή της _read_test, όπου ελέγχει αν υπάρχουν ενεργοί γραφείς (νήματα από την _write_test), αν το πρόγραμμα εκτελεστεί απευθείας με την εντολή read x (όπου x, ο αριθμός αναγνώσεων, πχ 100000), δεν θα μπει στην while και θα συνεχίσει η κανονική εκτέλεση του προγράμματος. Αν όμως εκτελεστεί η readwrite x y (όπου x το συνολικό πλήθος καταχωρίσεων, και y το ποσοστό εγγραφής), τότε λόγω των γραφέων που θα είναι ενεργοί, τα νήματα ανάγνωσης θα περιμένουν signal/broadcast για να ξεκινήσουν την εκτέλεσή τους (kiwi.c, γραμμές 123-127).

```
pthread_mutex_lock(&writeMutex);                        // Κλειδώνουμε
while (writeFlag > 0) {                                 // Όσο υπάρχουν ενεργοί γραφείς
        pthread_cond_wait(&writeDone, &writeMutex);     // Περίμενε signal/broadcast
}                                                       //
pthread_mutex_unlock(&writeMutex);                      // Όταν το πάρει, ξεκλειδώνουμε
```

Παράδειγμα εκτέλεσης

Για `./kiwi-bench read 100000`

Στο bench.c μπαίνει στην συνθήκη read, όπου το 100000, το σπάμε με βάση τον αριθμό των διαθέσιμων νημάτων (μπορεί να τροποποιηθεί από το αρχείο bench.h, γραμμή 16, έχουμε βάλει 16 νήματα εμείς). Κάθε νήμα που δημιουργείται θα έχει τον δικό του φόρτο για ανάγνωση. Κάθε νήμα αρχικοποιεί τις μεταβλητές του και αυξάνει τον threadCounter, το 1ο νήμα θα ανοίξει την βάση. Μπαίνουν στην for loop με το range που έχουν να αναζητήσουν,

ψάχνουν τα κλειδιά καλώντας την db_get (αν τα βρουν αυξάνουν το found τους), όταν τελειώσουν μειώνουν τον threadCounter και το τελευταίο νήμα κλείνει την βάση. Το κάθε νήμα τυπώνει τα ατομικά στατιστικά της εκτέλεσης του στο αρχείο output.txt και στο τερματικό τα αρχικά μηνύματα που υπήρχαν στον κώδικα. (Υπάρχει στην αρχή του κώδικα συνθήκη ελέγχου για ενεργούς γραφείς, εδώ δεν μας απασχολεί, θα αναλυθεί στην readwrite).

Η λειτουργία της πολυνηματικής υλοποίησης για την write

Η βασική ιδέα της υλοποίησης μας είναι πως για όσες καταχωρίσεις θέλουμε να γράψουμε (πχ, 100), σπάμε τον φόρτο εργασίας στον αριθμό των threads που δημιουργούμε, πχ για 4 threads και 100 καταχωρίσεις, ο φόρτος εργασίας του κάθε νήματος θα είναι 25, και κάθε νήμα θα έχει να διαβάσει από ένα συγκεκριμένο range
[0-25), [25-50), [50-75), [75-100).

Τεκμηρίωση Κώδικα

Για αυτόν τον λόγο, τροποποιήσαμε την write κατάλληλα ώστε να μπορεί να δέχεται πολλαπλά ορίσματα μέσα από ένα struct (bench.h, γραμμές 23-28).

```
struct data {                                   //
        long int count;                         // Ο συνολικός αριθμός καταχωρίσεων
        int r;                                  // Παράμετρος για εγγραφή άγνωστων κλειδιών
        long int index_start, index_end;        // Το διάστημα που θα γράφει το κάθε νήμα
        pthread_t       tid;                    // Το αναγνωριστικό του κάθε νήματος
};                                              //
```

Για τον αριθμό των νημάτων, υπάρχει η numOfThreads (bench.h, γραμμή 16), η οποία έχει γίνει define και μπορεί να αλλάξει.

```
#define numOfThreads 16                         // Αριθμός νημάτων για read και write
```

Η main, όταν κληθεί με το όρισμα write, θα εκτελέσει κάποιες βασικές εντολές και μετά έχουν γίνει κάποιες προσθήκες (bench.c, γραμμές 90-110) για την σωστή λειτουργία της πολυνηματικής εγγραφής.

```
struct data thread_args[numOfThreads];                  // Φτιάχνουμε μία λίστα από struct data
long int index_n = count/numOfThreads;                  // Ο φόρτος για το κάθε νήμα
int remainder = count % numOfThreads;                   // Το υπόλοιπο της παραπάνω ακέραιας διαίρεσης
long int index_start = 0;                               // Η αρχική τιμή του 1ου διαστήματος εγγραφής
long int index_end = index_start + index_n;             // Η τελική τιμή του 1ου διαστήματος εγγραφής
for (int i = 0; i < numOfThreads; i++) {                // Για όσα threads έχουμε
        thread_args[i].count = count;                   // Περνάμε το count σαν παράμετρο
        thread_args[i].r = r;                           // Περνάμε το r σαν παράμετρο
        thread_args[i].index_start = index_start;       // Περνάμε την αρχή του διαστήματος εγγραφής σαν παράμετρο
        thread_args[i].index_end = index_end;           // Περνάμε το τέλος του διαστήματος εγγραφής σαν παράμετρο
        pthread_create(&thread_args[i].tid, NULL, _write_test, (void *) &thread_args[i]); // Δημιουργούμε το νέο νήμα
        index_start = index_end;                        // Αλλάζουμε τα διαστήματα, τέλος παλιού -> αρχή νέου
        if (i == numOfThreads-2) {                      // Αν το επόμενο είναι το τελευταίο νήμα
                index_end = index_end + index_n + remainder;     // Λάβε υπόψιν το υπόλοιπο που μπορεί να έχει μείνει
        } else {                                        // Αλλιώς
                index_end = index_end + index_n;        // Το τέλος του νέου διαστ., θα είναι το παλιό τέλος +φόρτος
        }                                               //
}                                                       //
for (int i = 0; i < numOfThreads; i++){                 // Μπλοκάρει την εκτέλεση
        pthread_join(thread_args[i].tid, NULL);         // μέχρι να επιστρέψουν όλα τα νήματα
}                                                       //
```

Καλούμε μέσα από τα threads, την _write_test (kiwi.c), εκεί έχουν γίνει οι εξής προσθήκες/αλλαγές.

Δηλώσεις mutexes, conditional variable και global μεταβλητών (kiwi.c, γραμμές 9-22).

```
// Δήλωση των mutex που θα χρησιμοποιηθούν
pthread_mutex_t writeStatsToFile = PTHREAD_MUTEX_INITIALIZER;     // Για την εγγραφή στατιστικών σε αρχείο
pthread_mutex_t dbLock = PTHREAD_MUTEX_INITIALIZER;               // (Για την _write_test) κλείδωμα της εγγραφής.
pthread_mutex_t dbOpenClose = PTHREAD_MUTEX_INITIALIZER;          // Για το άνοιγμα και το κλείσιμο της DB

pthread_mutex_t writeMutex = PTHREAD_MUTEX_INITIALIZER;           // (για την readwrite) κλειδαριά για τους αναγνώστες
pthread_cond_t writeDone = PTHREAD_COND_INITIALIZER;              // (για την readwrite) μεταβλητή συνθήκης


FILE *fp;                       // Δήλωση δείκτη σε αρχείο

DB* db;                         // Δήλωση της βάσης (από τοπική σε global), για κοινή πρόσβαση από όλα τα νήματα
int threadCounter = 0;          // Για το άνοιγμα και το κλείσιμο της βάσης
int writeFlag = 0;              // (readwrite) Μεταβλητή που μετρά τους ενεργούς γραφείς, για να δούμε αν τελείωσαν
```

Κάθε νήμα εγγραφής, κλειδώνει, αυξάνει τον μετρητή writeFlag και ξεκλειδώνει (kiwi.c, γραμμές 26-28).

```
pthread_mutex_lock(&writeMutex);        // Κλειδώνουμε
writeFlag++;                            // Αυξάνουμε τον μετρητή
pthread_mutex_unlock(&writeMutex);      // Ξεκλειδώνουμε
```

Περνάμε σε τοπικές μεταβλητές τις παραμέτρους count και r (kiwi.c, γραμμές 30-32).

```
struct data *d = (struct data *) arg;           // Παίρνουμε τις παραμέτρους και τις κάνουμε cast σε struct data
long int count = d->count;                      // Η τοπική count, παίρνει την τιμή count από τις παραμέτρους
int r = d->r;                                   // Αντίστοιχα και η r
```

Ανοίγουμε την βάση (μία φορά) (kiwi.c, γραμμές 48-53).

```
pthread_mutex_lock(&dbOpenClose);               // Κλειδώνουμε
threadCounter++;                                // Αυξάνουμε τον μετρητή threadCounter
if (threadCounter == 1) {                       // Αν είναι το 1ο νήμα, τότε
        db = db_open(DATAS);                    // ανοίγει την βάση
}                                               //
pthread_mutex_unlock(&dbOpenClose);             // Ξεκλειδώνουμε
```

Το κάθε νήμα εκτελεί μία for loop, με αρχή και τέλος από τα ορίσματα που περάσαμε σε κάθε νήμα (kiwi.c, γρ. 56).

```
// πχ για write 100 με 4 νήματα, κάθε νήμα θα έχει διαφορετικό διάστημα, [0-25), [25-50), [50-75) και [75-100)
for (i = d->index_start; i < d->index_end; i++) {
```

Κάθε νήμα κλειδώνει την βάση, προσθέτει το κλειδί που θέλει με την db_add και ξεκλειδώνει (kiwi.c, γρ. 70-72).

```
pthread_mutex_lock(&dbLock);                    // Κλειδώνουμε
db_add(db, &sk, &sv);                           // Προσθέτουμε ένα νέο κλειδί
pthread_mutex_unlock(&dbLock);                  // Ξεκλειδώνουμε
```

Κλείνουμε την βάση (μία φορά) (kiwi.c, γραμμές 84-89).

```
pthread_mutex_lock(&dbOpenClose);               // Κλειδώνουμε
threadCounter--;                                // Μειώνουμε τον μετρητή threadCounter
if (threadCounter == 0) {                       // Αν είναι το τελευταίο νήμα, τότε
        db_close(db);                           // κλείνει την βάση
}                                               //
pthread_mutex_unlock(&dbOpenClose);             // Ξεκλειδώνουμε
```

Για την readwrite (δεν επηρεάζει την εκτέλεση της απλής write) (kiwi.c, γραμμές 93-98).

```
pthread_mutex_lock(&writeMutex);                // Κλειδώνουμε
writeFlag--;                                    // Μειώνουμε το writeFlag κατά ένα
if (writeFlag == 0) {                           // και αν φτάσει στο 0
        pthread_cond_broadcast(&writeDone);     // Στέλνουμε σήμα σε όλους τους αναγνώστες για να ξεκινήσουν
}                                               //
pthread_mutex_unlock(&writeMutex);              // Ξεκλειδώνουμε
```

Γράφουμε τα στατιστικά της εκτέλεσης για κάθε νήμα στο αρχείο output.txt (kiwi.c, γραμμές 105-111).

```
// Γράφουμε σε ένα αρχείο, τα στατιστικά της εγγραφής
pthread_mutex_lock(&writeStatsToFile);   // Κλειδώνουμε
fp = fopen("output.txt","a+");           // Ανοίγουμε το αρχείο (αν δεν υπάρχει δημιουργείται)
long int threadTotal = d->index_end-d->index_start;      // Υπολογίζουμε το συνολικό φόρτο του κάθε νήματος

// Τυπώνουμε στο αρχείο τα στατιστικά εκτέλεσης του νήματος
fprintf(fp, "Thread %ld completed execution, %ld writes: %.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec);\n"
,d->tid, threadTotal, (double) (cost/threadTotal), (double) (threadTotal/cost), cost);

fclose(fp);                              // Κλείνουμε το αρχείο
pthread_mutex_unlock(&writeStatsToFile); // Ξεκλειδώνουμε
```

Παράδειγμα εκτέλεσης

Για `./kiwi-bench write 100000`

Στο bench.c μπαίνει στην συνθήκη write, όπου το 100000, το σπάμε με βάση τον αριθμό των διαθέσιμων νημάτων (μπορεί να τροποποιηθεί από το αρχείο bench.h, γραμμή 16, έχουμε βάλει 16 νήματα εμείς). Κάθε νήμα που δημιουργείται θα έχει τον δικό του φόρτο για εγγραφή. Κάθε νήμα αυξάνει το writeFlag, αρχικοποιεί τις μεταβλητές του και αυξάνει τον threadCounter, το 1ο νήμα θα ανοίξει την βάση. Μπαίνουν στην for loop με το range που έχουν να εγγράψουν, προσθέτουν τα κλειδιά καλώντας την db_add, όταν τελειώσουν μειώνουν τον threadCounter και το τελευταίο νήμα κλείνει την βάση και μειώνουν το writeFlag, και όταν φτάσει στο μηδέν (δεν υπάρχουν ενεργοί γραφείς), στέλνει σήμα για να ξεκινήσουν όλοι οι αναγνώστες (δεν μας απασχολεί στην απλή write, θα αναλυθεί στην readwrite). Το κάθε νήμα τυπώνει τα ατομικά στατιστικά της εκτέλεσης του στο αρχείο output.txt και στο τερματικό τα αρχικά μηνύματα που υπήρχαν στον κώδικα.

Η λειτουργία της πολυνηματικής υλοποίησης για την readwrite

Η βασική ιδέα της υλοποίησης μας είναι πως για όσες καταχωρίσεις θέλουμε συνολικά (πχ, 1000) και ανάλογα το ποσοστό των καταχωρήσεων που θέλουμε να είναι εγγραφές (πχ, 75), τότε θα κάνουμε 750 εγγραφές και 250 αναγνώσεις, για τις εγγραφές θα χρησιμοποιήσουμε 75 threads εγγραφής (ίσα με το ποσοστό εγγραφής που μας δίνεται) και για τις αναγνώσεις θα έχουμε 25 threads ανάγνωσης. Ο φόρτος για τις εγγραφές θα σπάσει ανάμεσα στα 75 νήματα, δηλαδή το κάθε νήμα εγγραφής θα έχει να κάνει 750/75 = 10 εγγραφές και αντίστοιχα, για τις αναγνώσεις θα έχουμε 250/25 = 10 αναγνώσεις για κάθε νήμα.

Τεκμηρίωση Κώδικα

Στην main, του bench.c, προσθέσαμε μία νέα επιλογή, για την readwrite (bench.c, γραμμές 146-206).

Ανάγνωση των παραμέτρων που δόθηκαν (bench.c, γραμμές 146-158).

```
} else if (strcmp(argv[1], "readwrite") == 0) {                          // Η νέα επιλογή, readwrite
        int r = 0;                                                       // --Από τον κώδικα των read και write--
        count = atoi(argv[2]);                                           // --
        _print_header(count);                                            // --
        _print_environment();                                            // --
        if (argc == 5)                                                   // Για random τιμές, θα έχουμε το 5ο όρισμα
                r = 1;                                                   // --
        int writePercentage = atoi(argv[3]);                             // Εδώ αποθηκεύουμε το ποσοστό των εγγραφών
        if (writePercentage < 1 || writePercentage > 99) {               // Έλεγχος, αν η τιμή που δόθηκε είναι έγκυρη
                printf("Write percentage should be in range [1-99]\n");  // Μήνυμα σφάλματος
                exit(-1);                                                // Τερματισμός προγράμματος
        }                                                                //
        int readPercentage = 100-writePercentage;                        // Αποθηκεύουμε το ποσοστό των αναγνώσεων
```

Φτιάχνουμε τα νέα νήματα για εγγραφή (bench.c, γραμμές 162-179). Ο κώδικας είναι ίδιος με την write με μόνη διαφορά την χρήση thread_args_write αντί για thread_args, διαφορετικό αριθμό για την index_n_write (θα διαβάσει ένα ποσοστό των καταχωρίσεων που δώσαμε) και remainder_write, και διαφορετικά ονόματα μεταβλητών. Η λειτουργικότητα παραμένει ίδια με την write και έχει τεκμηριωθεί παραπάνω.

```
struct data thread_args_write[writePercentage];
long int index_n_write = ((count*writePercentage)/100)/writePercentage;
int remainder_write = ((count*writePercentage)/100) % writePercentage;
long int index_start_write = 0;
long int index_end_write = index_start_write + index_n_write;
for (int i = 0; i < writePercentage; i++) {
        thread_args_write[i].count = ((count*writePercentage)/100);
        thread_args_write[i].r = r;
        thread_args_write[i].index_start = index_start_write;
        thread_args_write[i].index_end = index_end_write;
        pthread_create(&thread_args_write[i].tid, NULL, _write_test, (void *) &thread_args_write[i]);
        index_start_write = index_end_write;
        if (i == writePercentage-2) {
                index_end_write = index_end_write + index_n_write + remainder_write;
        } else {
                index_end_write = index_end_write + index_n_write;
        }
}
```

Φτιάχνουμε τα νέα νήματα για ανάγνωση (bench.c, γραμμές 182-199). Ο κώδικας είναι ίδιος με την read με μόνη διαφορά την χρήση thread_args_read αντί για thread_args, διαφορετικό αριθμό για την index_n_read (θα διαβάσει ένα ποσοστό των καταχωρίσεων που δώσαμε) και remainder_read, και διαφορετικά ονόματα μεταβλητών. Η λειτουργικότητα παραμένει ίδια με την read και έχει τεκμηριωθεί παραπάνω.

```
struct data thread_args_read[readPercentage];
long int index_n_read = ((count*readPercentage)/100)/readPercentage;
int remainder_read = ((count*readPercentage)/100) % readPercentage;
long int index_start_read = 0;
long int index_end_read = index_start_read + index_n_read;
for (int i = 0; i < readPercentage; i++) {
        thread_args_read[i].count = ((count*readPercentage)/100);
        thread_args_read[i].r = r;
        thread_args_read[i].index_start = index_start_read;
        thread_args_read[i].index_end = index_end_read;
        pthread_create(&thread_args_read[i].tid, NULL, _read_test, (void *) &thread_args_read[i]);
        index_start_read = index_end_read;
        if (i == readPercentage-2) {
                index_end_read = index_end_read + index_n_read + remainder_read;
        } else {
                index_end_read = index_end_read + index_n_read;
        }
}
```

Μπλοκάρουμε και περιμένουμε τα νήματα να τελειώσουν (bench.c, γραμμές 201-206). Ο κώδικας έχει τεκμηριωθεί στα αντίστοιχα τμήματα των read και write.

```
for (int i = 0; i < writePercentage; i++){
        pthread_join(thread_args_write[i].tid, NULL);
}
for (int i = 0; i < readPercentage; i++){
        pthread_join(thread_args_read[i].tid, NULL);
}
```

Καλούμε τις _write_test και _read_test. Τώρα με χρήση conditional variable (της writeDone, kiwi.c, γραμμή 15), φροντίζουμε να εκτελεστούν πρώτα οι γραφείς και μετά οι αναγνώστες. Συγκεκριμένα, μόλις κληθούν οι γραφείς, αυξάνουν έναν global μετρητή, τον writeFlag, και όσο είναι ενεργοί οι γραφείς, το writeFlag θα είναι > 0 (kiwi.c, γραμμές 26-28). Οι αναγνώστες που θα κληθούν στη συνέχεια, ελέγχουν αν υπάρχουν ενεργοί γραφείς (kiwi.c, γραμμές 123-127), αν υπάρχουν (δηλαδή writeFlag > 0) τότε, σταματούν την εκτέλεση και περιμένουν σήμα από τους γραφείς πως έχουν τελειώσει (kiwi.c, γραμμές 93-98).

Παράδειγμα εκτέλεσης

Για `./kiwi-bench readwrite 100000 64`

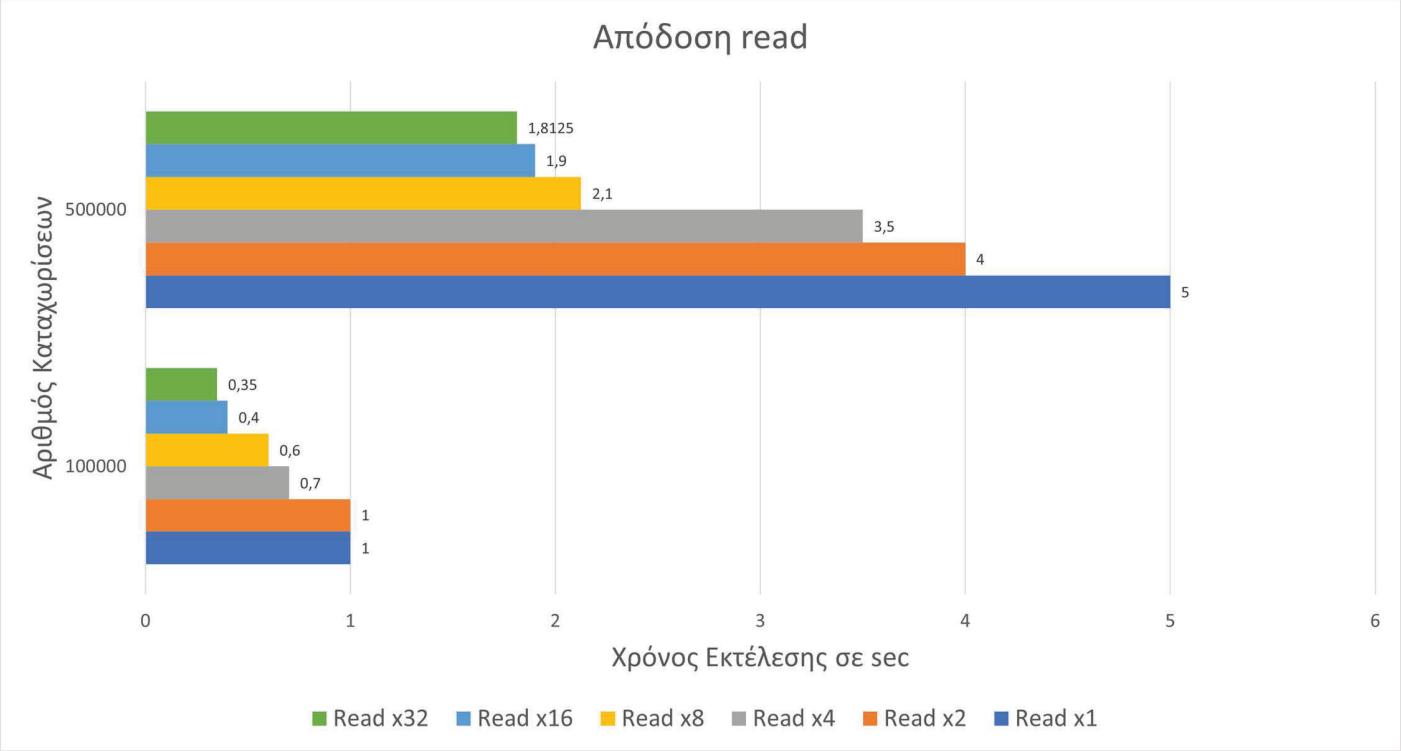
(Γενικά: `./kiwi-bench readwrite <συνολικός αριθμός καταχωρίσεων> <ποσοστό εγγραφών> <random>`)

Στο bench.c μπαίνει στην συνθήκη readwrite, όπου θα έχουμε 64% εγγραφές και 36% αναγνώσεις, δηλαδή θα κάνουμε 64000 εγγραφές και 36000 αναγνώσεις. Θα δημιουργήσουμε 100 νήματα συνολικά, 64 που θα κάνουν εγγραφές και 36 που θα κάνουν αναγνώσεις. Κάθε νήμα εγγραφής θα έχει φόρτο ((100000*64)/100)/64 = 1000 και αντίστοιχα για κάθε νήμα ανάγνωσης ((100000*36)/100)/36 = 1000. Τα νήματα εγγραφής αυξάνουν το writeFlag ώστε να μπλοκάρουν τα νήματα ανάγνωσης (λόγω του ότι writeFlag>0). Η λειτουργία της _write_test εκτελείται ακριβώς όπως έχει περιγραφεί προηγουμένως στο παράδειγμα εκτέλεσης της write. Όταν τελειώσουν όλοι οι γραφείς, ο τελευταίος, στέλνει σήμα σε όλους τους αναγνώστες ώστε να ξεκινήσουν την ανάγνωση (αφού πλέον writeFlag==0). Η λειτουργία της _read_test εκτελείται ακριβώς όπως έχει περιγραφεί προηγουμένως στο παράδειγμα εκτέλεσης της read.

Στατιστικά

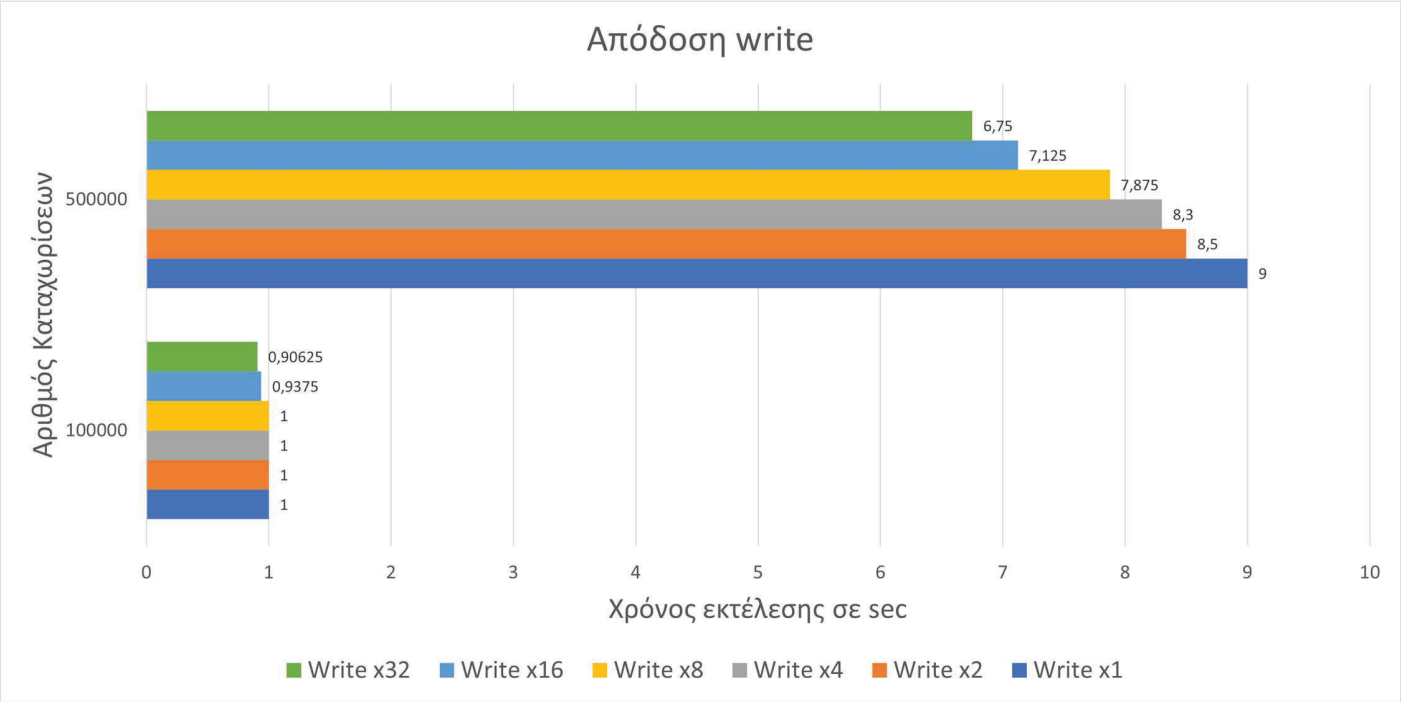
Ρυθμίσεις Εικονικής Μηχανής: 4 Πυρήνες (AMD Ryzen 7 4800U) και 4GB RAM

Εκτέλεση της read για διαφορετικό πλήθος νημάτων και καταχωρήσεων



Παρατηρούμε πως όσο περισσότερα είναι τα νήματα τόσο ταχύτερη είναι η ανάγνωση (καταχωρήσεις 1000 και 10000 έδιναν χρόνο 0, και δεν συμπεριλήφθηκαν).

Εκτέλεση της write για διαφορετικό πλήθος νημάτων και καταχωρήσεων



Αντίστοιχα και για τις εγγραφές, παρατηρούμε πως όσο περισσότερα είναι τα νήματα τόσο ταχύτερη είναι η εγγραφή (καταχωρήσεις 1000 και 10000 έδιναν χρόνο 0, και δεν συμπεριλήφθηκαν).

Στατιστικά στο αρχείο output.txt

./kiwi-bench write 100

Thread 140325275637504 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325267244800 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325300815616 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325258852096 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325242066688 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325104383744 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325200103168 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325183317760 completed execution, 10 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325225281280 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325250459392 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325292422912 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325284030208 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325233673984 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325216888576 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325309208320 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);
Thread 140325208495872 completed execution, 6 writes: 0.000000 sec/op; inf writes/sec(estimated); cost:0.000(sec);

./kiwi-bench read 100

Thread 139720441702144 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720453519104 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720609490688 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720424916736 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720592705280 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720416524032 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720617883392 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720550741760 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720575919872 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720542349056 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720601097984 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720559134464 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720533956352 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720584312576 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720433309440 completed execution, found 6/6 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);
Thread 139720408131328 completed execution, found 10/10 reads: 0.000000 sec/op; inf reads/sec(estimated); cost:0.000(sec);

Output της make all

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
make[1]: Nothing to be done for 'all'.
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
kiwi.c: In function ‘_write_test’:
kiwi.c:119:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
kiwi.c: In function ‘_read_test’:
kiwi.c:209:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$
```