

Εργαστήριο 1: Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

Ευστάθιος Ανδρεόπουλος AM: 4630

Γιώργος Χατζηλίγος AM: 4835

CONTENTS

Περιγραφή μηχανής kiwi	2
Γενική περιγραφή του προβλήματος.....	3
Περιγραφή της λύσης του προβλήματος.....	4
Bench.c και περιγραφή αλλαγών - τροποποιήσεων.....	5
Bench.h και περιγραφή αλλαγών – τροποποιήσεων	16
Kiwi.c και περιγραφή αλλαγών - τροποποιήσεων.....	17
Db.h και περιγραφή αλλαγών – τροποποιήσεων	21
Db.c και περιγραφή αλλαγών – τροποποιήσεων	22
Test	28

ΠΕΡΙΓΡΑΦΗ ΜΗΧΑΝΗΣ KIWI

Το Kiwi είναι μια μηχανή αποθήκευσης ανοιχτού κώδικα για καταστήματα βασικών τιμών που βασίζεται στη δομή δεδομένων Log-Structured Merge Tree (LSM tree). Ακολουθούν ορισμένα βασικά χαρακτηριστικά και πληροφορίες για το ακτινίδιο:

- Δεντρικό δέντρο LSM: Το δέντρο LSM είναι μια δομή δεδομένων που βασίζεται σε δίσκο και είναι βελτιστοποιημένη για βαρύ φόρτο εργασίας. Αποτελείται από πολλαπλά επίπεδα, όπου τα δεδομένα γράφονται αρχικά σε ένα buffer στη μνήμη, που ονομάζεται memtable. Μόλις το memtable γεμίσει, τα περιεχόμενά του ξεπλένονται στο δίσκο και γίνονται το κάτω επίπεδο του δέντρου LSM. Καθώς γράφονται περισσότερα δεδομένα, δημιουργούνται νέα επίπεδα στο δίσκο, με κάθε επίπεδο να είναι διαδοχικά μεγαλύτερο και να περιέχει πιο συμπαγή δεδομένα.
- Συμπίεση: Το ακτινίδιο χρησιμοποιεί μια τεχνική που ονομάζεται ισοπεδωμένη συμπίεση για να συγχωνεύσει δεδομένα από τα χαμηλότερα επίπεδα του δέντρου LSM σε υψηλότερα επίπεδα. Αυτό βοηθά στη μείωση του αριθμού των αναζητήσεων δίσκου που απαιτούνται για την ανάγνωση ενός ζεύγους κλειδιού-τιμής και βελτιώνει την απόδοση ανάγνωσης.
- Ευρετήριο B-tree: Το Kiwi χρησιμοποιεί έναν δείκτη B-tree για να εντοπίσει δεδομένα στο δέντρο LSM. Αυτό βοηθά στη μείωση του αριθμού των αναζητήσεων δίσκου που απαιτούνται για την εύρεση ενός συγκεκριμένου ζεύγους κλειδιού-τιμής.
- Ενίσχυση εγγραφής: Το ακτινίδιο στοχεύει να ελαχιστοποιήσει την ενίσχυση εγγραφής μειώνοντας τον όγκο των δεδομένων που πρέπει να ξαναγραφτούν κατά τη συμπύκνωση.
- Ερωτήματα εύρους: Το Kiwi υποστηρίζει ερωτήματα εύρους, τα οποία μπορούν να εκτελεστούν αποτελεσματικά σαρώνοντας τα επίπεδα δέντρων LSM με τη σειρά.
- Συντονίσιμες παράμετροι: Το Kiwi επιτρέπει στους χρήστες να συντονίζουν διάφορες παραμέτρους, όπως το μέγεθος του memtable, τον αριθμό των επιπέδων στο δέντρο LSM και τη στρατηγική συμπίεσης.
- Συνολικά, το Kiwi είναι μια σύγχρονη και αποδοτική μηχανή αποθήκευσης που είναι κατάλληλη για μεγάλο φόρτο εργασίας και ερωτήματα εύρους.

ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Το πρόβλημα αφορά μια μηχανή αναζήτησης η οποία λειτουργεί με 2 νήματα-threads και έχει ως κύριο σκοπό την εξυπηρέτηση ενός ή πολλών χρηστών. Βασικό πρόβλημα αποτελεί το διάβασμα και το γράψιμο σε μια βάση δεδομένων. Το πρόγραμμα έχει τη λειτουργία "put", η οποία γράφει στη βάση δεδομένων νέα στοιχεία είτε στο memtable είτε στο sst.

Υπάρχουν δύο νήματα στη μηχανή αποθήκευσης. Το πρώτο νήμα εκτελεί μια σειρά λειτουργιών put ή get μία μετά την άλλη, όπως ορίζεται από τη γραμμή εντολών. Το δεύτερο νήμα έχει ως αρμοδιότητα τη συμπίεση των αρχείων όταν πληρούνται οι απαιτούμενες συνθήκες. Όταν δηλαδή το memtable μετακινείται στον δίσκο, συγχωνεύεται με τα αρχεία (στο επίπεδο 0 ή 1) των οποίων τα κλειδιά επικαλύπτονται με τα κλειδιά του memtable. Επιπρόσθετα, ενεργοποιείται σύμπτυξη αρχείων (compaction) όταν το πλήθος των αρχείων στο επίπεδο 0 ξεπερνά ένα προκαθορισμένο κατώφλι (π.χ., 4), ή το συνολικό μέγεθος των αρχείων σε ένα επίπεδο ξεπερνάει ένα προκαθορισμένο κατώφλι για το επίπεδο αυτό. Με την σύμπτυξη, τα αρχεία ενός επιπέδου συγχωνεύονται σε ένα αρχείο που προστίθεται στο επόμενο επίπεδο.

Η λειτουργία put επιδρά στο memtable για να εισάγει ένα νέο στοιχείο στη δομή και, αν απαιτείται, συγχωνεύει το τρέχον memtable με ένα αρχείο sst, που μπορεί να οδηγήσει σε περαιτέρω σύμπτυξη από το ένα επίπεδο στο επόμενο. Αντίστοιχα, η λειτουργία get αναζητά πρώτα στο memtable για το διθέν κλειδί και, αν δεν βρεθεί εκεί, συνεχίζει στα αρχεία sst ξεκινώντας από το επίπεδο 0 και πηγαίνοντας σε επόμενα επίπεδα όπως απαιτείται. Αν έχει δρομολογηθεί η συγχώνευση ενός memtable, αυτό πρέπει επίσης να αναζητηθεί πριν τα αρχεία sst. Η αναζήτηση σε ένα αρχείο sst γίνεται γρηγορότερα με τη χρήση ενός Bloom filter που περιέχει τους κατακερματισμούς των κλειδιών που έχουν ήδη εισαχθεί στο συγκεκριμένο αρχείο.

ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΥΣΗΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ

Έχουμε μια μηχανή αναζήτησης η οποία θέλουμε οι λειτουργίες της να γίνονται πολυνηματικά. Στην μηχανή αναζήτησης kiwi έχουμε αρχικά 2 threads τα οποία περιγράψαμε αρχικά ποιες είναι λειτουργίες τους . Τώρα θέλουμε να έχουμε πολλά thread που να εκτελούν είτε read είτε write είτε ταυτόχρονα readwrite στην βάση δεδομένων .

Η bench.c και η kiwi.c είναι 2 αρχεία της γλώσσας τα οποία του front-end τα οποία μεταξύ τους συνεργάζονται το . Το bench.c έχει ως κύριο σκοπό του να πάρει τα δεδομένα από τον χρήστη και κάποια δεδομένα από τον υπολογιστή του χρήστη .Η kiwi.c με την σειρά της έχει ως κύριο σκοπό να παίρνει τα δεδομένα από την bench.c του χρήστη και να επικοινωνήσει με το backend δηλαδή την db.c και db.h ώστε να εκτελέσει τις λειτουργίες read και write.

Η bench.c οπότε πρέπει να φτιάχνει ένα πίνακα με κωδικούς νημάτων pthread_t *id_num και ανάλογα με το πόσα συνολικά read ή write έχουμε και thread να τα περνάει με μια struct στο bench.h ώστε να μπορεί να έχει πρόσβαση και το kiwi.c . Ανάλογα τα πόσα threads έχουμε τόσα write ή read θα πρέπει να εκτελέσουμε και αυτό θα γίνει με την συνάρτηση pthread_create(). Αφού κατασκευάσουμε τα νήματα κάνουμε και πρέπει να περιμένουμε και τα υπόλοιπα νήματα να τελειώσουν (Κλασσική μεθοδολογία για την κατασκευή νημάτων) . Έπειτα να πληροφορήσουμε το χρήστη με γενικές επιδόσεις της πολυνηματικής εκτέλεσης.

Η bench.h χρησιμεύει σαν διεπαφή μεταξύ του kiwi.c και του bench.c ώστε να περνάμε μεταβλητές όπως τα νήματα και το πόσες λειτουργίες με αυτά θέλουμε να κάνουμε . Επίσης είναι σημαντικό να αναφέρουμε εδώ και το τις μεταβλητές τύπου pthread_mutex_t που χρησιμεύουν για να κάνουμε lock και unlock μέσα στην kiwi.c και να κάνουμε read και write .

Η kiwi.c θα επικοινωνεί με την βάση δεδομένων ανάλογα με το λόγο λειτουργιών/νήματα και θα εκτελεί είτε read είτε write. Επίσης χρησιμοποιούμε κλειδαριές που έχουν σκοπό να κλειδώνουν τα mutex που αφορούν την μεταβλητή writes και τα reads ώστε να μην δίνεται πρόσβαση και να υπάρχει αμοιβαίος αποκλεισμός μεταξύ των νημάτων .

BENCH.C ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΑΛΛΑΓΩΝ - ΤΡΟΠΟΠΟΙΗΣΕΩΝ

```
1 #include "bench.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #include <unistd.h>
6 #define R_RATE 10
7 #define W_RATE 90
8
```

Γραμμή 2^η : Το stdio.h είναι ένα αρχείο κεφαλίδας που έχει τις απαραίτητες πληροφορίες για να συμπεριλάβει τις συναρτήσεις εισόδου/εξόδου στο πρόγραμμά μας. Παράδειγμα printf, scanf κ.λπ.

Γραμμή 3^η : Το stdlib.h είναι ένα αρχείο κεφαλίδας που δηλώνει διάφορες συναρτήσεις βοηθητικού προγράμματος για μετατροπές τύπων, εκχώρηση μνήμης, αλγόριθμους και άλλες παρόμοιες περιπτώσεις χρήσης. Παράδειγμα malloc, calloc κ.λπ.

Γραμμή 5^η : Η κεφαλίδα <unistd.h> ορίζει διάφορες συμβολικές σταθερές και τύπους και δηλώνει διάφορες συναρτήσεις.

Γραμμή 6^η : Η οδηγία #define χρησιμοποιείται για να δηλώσει ορισμένες σταθερές τιμές ή μια έκφραση με όνομα που μπορεί να χρησιμοποιηθεί σε όλο το πρόγραμμά μας C.

```
78 // This function takes a void pointer argument named values
79 void *write_request(void *values){
80     // Cast the void pointer "values" to a pointer of type "struct data"
81     struct data *d = (struct data *)values;
82     // Call the function "_write_test" with the arguments specified in the "struct data" pointer "d"
83     _write_test(
84         d->number_count,      // Number of values to be written
85         d->number_r,          // Random seed value
86         d->number_threads     // Number of threads to use
87     );
88     // Return 0 as the function's exit status
89     return 0;    // Return 0 as the function's exit status
90 }
91
92
93 // This function takes a void pointer argument named values
94 void *read_request(void *values){
95     // Cast the void pointer "values" to a pointer of type "struct data"
96     struct data *d = (struct data *)values;
97     // Call the function "_write_test" with the arguments specified in the "struct data" pointer "d"
98     _read_test(
99         d->number_count,      // Number of values to be written
100        d->number_r,          // Random seed value
101        d->number_threads     // Number of threads to use
102    );
103    // Return 0 as the function's exit status
104    return 0;    // Return 0 as the function's exit status
105 }
106
```

Η συνάρτηση write_request(Γραμμές 78-91) είναι μια συνάρτηση που ορίζει ένα νήμα (thread) που θα κληθεί αργότερα από την κύρια συνάρτηση(main). Η συνάρτηση αυτή λαμβάνει ως είσοδο έναν δείκτη σε μια δομή data. Αυτή η δομή περιέχει τρία ακέραιους αριθμούς, τον αριθμό count που πρέπει να γραφτούν, τον αριθμό r(random) και τον αριθμό των νημάτων(threads). Μετά το casting του δείκτη στην δομή data, η συνάρτηση καλεί τη συνάρτηση _write_test με τους παραμέτρους number_count, number_r και number_threads που αντλούνται από τη δομή data.

Η συνάρτηση read_request(Γραμμές 93-105) είναι μια συνάρτηση που ορίζει ένα νήμα (thread) που θα κληθεί αργότερα από την κύρια συνάρτηση(main).

Η συνάρτηση αυτή λαμβάνει ως είσοδο έναν δείκτη σε μια δομή data. Αυτή η δομή περιέχει τρία ακέραιους αριθμούς, τον αριθμό count που πρέπει να γραφτούν, τον αριθμό r(random) και τον αριθμό των νημάτων(threads).

Μετά το casting του δείκτη στην δομή data, η συνάρτηση καλεί τη συνάρτηση _read_test με τους παραμέτρους number_count, number_r και number_threads που αντλούνται από τη δομή data.

```
109 // Function to generate a random integer value greater than or equal to 50
110 int random_generator(){
111     // Generate a random integer value using the rand() function from the standard library
112     int random_value = rand();
113     // Initialize a counter variable to keep track of the number of iterations
114     int count = 0;
115     // While the random value is less than 50, generate a new random value
116     while (random_value < 50 ){
117         random_value = rand(); // Generate a new random value
118         fprintf(stderr, "\n [%d] random value : %d",count,random_value); // Print the current random value (optional)
119         count++; // Increment the counter variable
120     }
121     // Multiply the random value by 10 and return the result
122     int return_value = random_value*10;
123     return return_value;
124 }
125
126 }
```

Η συνάρτηση(Γραμμές 110-125) ξεκινά δημιουργώντας μια τυχαία ακέραια τιμή χρησιμοποιώντας τη συνάρτηση rand() και αποθηκεύοντάς την στη μεταβλητή random_value.

Στη συνέχεια, η συνάρτηση προετοιμάζει μια μεταβλητή μετρητή που ονομάζεται count για να παρακολουθεί τον αριθμό των επαναλήψεων.

Ο βρόχος while ξεκινά ελέγχοντας εάν η τιμή random_value είναι μικρότερη από 50. Εάν είναι, ο βρόχος δημιουργεί μια νέα τυχαία τιμή χρησιμοποιώντας τη rand() και την εκχωρεί στην τιμή random_value. Η πρόταση fprintf εντός του βρόχου είναι προαιρετική και χρησιμοποιείται για την εκτύπωση της τρέχουσας τυχαίας τιμής στην τυπική ροή σφαλμάτων (stderr). Ο βρόχος συνεχίζεται έως ότου η τιμή_τυχαίας τιμής είναι μεγαλύτερη ή ίση με 50.

Μετά την ολοκλήρωση του βρόχου, η συνάρτηση πολλαπλασιάζει την τιμή random_value επί 10 και επιστρέφει το αποτέλεσμα.

```

129 // Function to print additional information to the terminal by reading from a text file
130 void printf_info_terminal_by_txt(){
131     system("cat /home/my601/kiwi_end/kiwi-source/engine/Information.txt"); // Execute a shell command to print the contents of a text file to the terminal
132 }
133
134 // Function to perform error checking on input parameters
135 void error_cather(char *command , char *count , char *threads , char *n){
136
137     // Convert the threads and count parameters from strings to integers
138     int thread_int = atoi(threads);
139     int count_int = atoi(count);
140
141     // Check if the command parameter is one of three valid values
142     if (strcmp(command , "write") !=0 && strcmp(command , "read")!=0 && strcmp(command,"readwrite")!=0){
143         // If the command is not valid, print an error message to stderr and call a function to print additional information to the terminal
144         fprintf(stderr,"Input error \n");
145         printf_info_terminal_by_txt();
146         exit(1);    // Exit the program with an error code of 1
147     }
148
149     // Check if the count parameter is Less than 0
150     if (count_int < 0){
151         // If the count is negative, print an error message to stderr and call a function to print additional information to the terminal
152         fprintf(stderr,"Unable to write in db negative amount of data !\n");
153         printf_info_terminal_by_txt();
154         exit(1);    // Exit the program with an error code of 1
155     }
156
157     // Check if the string "n" can be converted to an integer 0 or 1
158     if ( atoi(n)!=0 && atoi(n)!=1){
159         // If "n" is not 0 or 1 , print an error message to standard error
160         fprintf(stderr,"Error in n!\n");
161         // Call a function to print additional information to the terminal.
162         printf_info_terminal_by_txt();
163         exit(1);    // Exit the program with an error code of 1
164     }
165 }
```

Η συνάρτηση(Γραμμές 129-132) printf_info_terminal_by_txt() εκτελεί μια εντολή φλοιού(shell) για να εκτυπώσει τα περιεχόμενα ενός αρχείου κειμένου στο τερματικό. Δεν λαμβάνει κάποια παράμετρο και δεν επιστρέφει κάποια τιμή.

Η συνάρτηση(Γραμμές 134-165) error_cather, είναι υπεύθυνη για τον έλεγχο σφαλμάτων στις παραμέτρους εισόδου που της μεταβιβάστηκαν. Η συνάρτηση παίρνει τέσσερις παραμέτρους, εντολή, count, νήματα και n, που είναι όλες δείκτες σε μεταβλητές τύπου char. Η συνάρτηση μετατρέπει πρώτα τα νήματα και τις παραμέτρους μέτρησης από συμβολοσειρές σε ακέραιους αριθμούς χρησιμοποιώντας τη συνάρτηση atoi. Στη συνέχεια, ελέγχει εάν η παράμετρος εντολής είναι μία από τις τρεις έγκυρες τιμές ("write", "read" ή "readwrite"). Εάν η εντολή δεν είναι έγκυρη, η συνάρτηση εκτυπώνει ένα μήνυμα σφάλματος στο stderr και βγαίνει από το πρόγραμμα με κωδικό σφάλματος 1. Στη συνέχεια, η συνάρτηση ελέγχει εάν η παράμετρος count είναι μικρότερη από 0. Εάν είναι, η συνάρτηση εκτυπώνει ένα μήνυμα σφάλματος στο stderr και βγαίνει από το πρόγραμμα με κωδικό σφάλματος 1. Τέλος, η συνάρτηση ελέγχει εάν η συμβολοσειρά που δείχνει η παράμετρος n μπορεί να μετατραπεί σε ακέραιο αριθμό 0 ή 1 χρησιμοποιώντας atoi. Εάν δεν μπορεί, η συνάρτηση εκτυπώνει ένα μήνυμα σφάλματος στο stderr και βγαίνει από το πρόγραμμα με κωδικό σφάλματος 1. Σε όλες τις περιπτώσεις, η συνάρτηση καλεί μια άλλη συνάρτηση που ονομάζεται printf_info_terminal_by_txt() για να εκτυπώσει πρόσθετες πληροφορίες στο τερματικό.

```

167 // This function prints the results of a random write benchmark test
168 // It takes in the number of the writes performed and the total time it took to perform them as input
169 void print_time_write(int number_count, double cost_of_writes){
170     // Print a horizontal line to make the output easier to read
171     printf(LINE);
172     // Prints the results of the write benchmark test.
173     // The output includes the number of writes performed, the time it took to perform one write operation,
174     // the estimated number of the writes per second, and the total time it took to perform all writes.
175     printf("|Random-Write    (done:%d): %.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec)\n",
176         ,number_count, (double)(cost_of_writes / number_count)
177         ,(double)(number_count / cost_of_writes)
178         ,cost_of_writes);
179 }
180
181 // This function prints the results of a random read benchmark
182 void print_time_read(int number_count , double cost_of_reads){
183     // Prints a horizontal line to make the output easier to read
184     printf(LINE);
185     // Prints the results of the read benchmark test
186     // The output includes the number of reads performed, the time it took to perform one read operation,
187     // the estimated number of the reads per second and the total time it took to perform all reads.
188     printf("|Random-Read    (done:%d): %.6f sec/op; %.1f reads /sec(estimated); cost:%.3f(sec)\n",
189         ,number_count, (double)(cost_of_reads / number_count),
190         ,(double)(number_count / cost_of_reads),
191         cost_of_reads);
192 }

```

Οι δύο συναρτήσεις print_time_write και print_time_read είναι υπεύθυνες για την εκτύπωση των αποτελεσμάτων μιας benchmark test τυχαίας write ή read, αντίστοιχα.

Η συνάρτηση print_time_write(Γραμμές 167-179) παίρνει δύο παραμέτρους: number_count, που αντιπροσωπεύει τον αριθμό των write που εκτελούνται και cost_of_write, που αντιπροσωπεύει τον συνολικό χρόνο που χρειάστηκε για την εκτέλεση των εγγραφών. Η συνάρτηση αρχικά εκτυπώνει μια οριζόντια γραμμή χρησιμοποιώντας τη μακροεντολή LINE που ορίζεται κάπου αλλού στον κώδικα και, στη συνέχεια, εκτυπώνει τα αποτελέσματα της benchmark test. Η έξοδος περιλαμβάνει τον αριθμό των write που πραγματοποιήθηκαν, το χρόνο που χρειάστηκε για να πραγματοποιηθεί μια λειτουργία write, τον εκτιμώμενο αριθμό write ανά δευτερόλεπτο και τον συνολικό χρόνο που χρειάστηκε για να πραγματοποιηθούν όλες οι write. Η συνάρτηση printf χρησιμοποιείται για την εκτύπωση αυτών των πληροφοριών στην κονσόλα.

Ομοίως, η συνάρτηση print_time_read(Γραμμές 181-192) λαμβάνει δύο παραμέτρους: number_count, που αντιπροσωπεύει τον αριθμό των read που πραγματοποιήθηκαν και cost_of_reads, που αντιπροσωπεύει τον συνολικό χρόνο που χρειάστηκε για να πραγματοποιηθούν οι reads. Η συνάρτηση εκτυπώνει μια οριζόντια γραμμή και, στη συνέχεια, εκτυπώνει τα αποτελέσματα της benchmark test. Η έξοδος περιλαμβάνει τον αριθμό των read που πραγματοποιήθηκαν, τον χρόνο που χρειάστηκε για να πραγματοποιηθεί μία λειτουργία read, τον εκτιμώμενο αριθμό read ανά δευτερόλεπτο και τον συνολικό χρόνο που χρειάστηκε για να πραγματοποιηθούν όλες οι read.

Συνολικά, αυτές οι λειτουργίες είναι χρήσιμες για την εκτύπωση των αποτελεσμάτων μιας benchmark test με σαφή και συνοπτικό τρόπο, διευκολύνοντας τον χρήστη να κατανοήσει την απόδοση του συστήματος που δοκιμάζεται.

```

194 // Function to read the read rate from the user
195 int read_function_rate(){
196     int read_rate ;
197     printf("Enter read rate :"); // Ask the user to enter the read rate
198     scanf("%d",&read_rate); // Read the read rate from the user
199     printf("\n");
200     // Check if the read rate is greater than 0
201     if (read_rate>0){
202         return read_rate; // If the read rate is valid, return it
203     }
204     printf("\nDefault read rate 90"); // If the read rate is not valid, print a default read rate message
205     return read_rate; // Return the read rate(even if it not valid)
206 }
207
208 //Function to read the write rate from the user
209 int write_function_rate(){
210     int write_rate ;
211     printf("Enter write rate :"); // Ask the user to enter the write rate
212     scanf("%d",&write_rate); // Read the write rate from the user
213     printf("\n");
214     // Check if the write rate is grater than 0
215     if (write_rate>0){
216         return write_rate; // If the write rate is valid, return it
217     }
218     printf("\nDefault write rate 10"); // If the write rate is not valid, print a default write rate message
219     return write_rate; // Return the write rate (even if it is not valid)
220 }
221

```

Αυτές οι δύο συναρτήσεις(Γραμμές 194-206 και 208-221) προτρέπουν τον χρήστη να εισάγει το ποσοστό ανάγνωσης και το ποσοστό εγγραφής, αντίστοιχα. Στη συνέχεια διαβάζουν τις τιμές που εισήγαγε ο χρήστης και πραγματοποιούν κάποια βασική επικύρωση εισόδου για να διασφαλίσουν ότι οι τιμές που έχουν εισαχθεί είναι μεγαλύτερες από το μηδέν. Εάν η εισαγόμενη τιμή είναι έγκυρη, η συνάρτηση επιστρέφει την τιμή. Εάν η εισαγόμενη τιμή δεν είναι έγκυρη, η συνάρτηση εκτυπώνει ένα προεπιλεγμένο μήνυμα(default) και επιστρέφει την τιμή που έχει εισαχθεί από τον χρήστη (η οποία μπορεί να είναι άκυρη).

Η χρήση για αυτές τις συναρτήσεις είναι να αποφύγουμε σφάλματα και να διασφαλίσουμε ότι το πρόγραμμα συμπεριφέρεται όπως αναμένεται.

```

223 // This function calculates the complement of the sum of the read_rate and write_rate and write_rate and return it.
224 // If read_rate or write_rate are negative, it prints an error message and exits the program
225 int complement_rate(int read_rate,int write_rate){
226     int complement ;
227     // Check if the read_rate is negative
228     if (read_rate<0){
229         // If the read_rate is negative, print an error message and exit the program
230         fprintf(stderr,"Unable to do multithreading use negative rate !\n");
231         printf_info_terminal_by_txt();
232         exit(1);    // Exit the program with an error code of 1
233     }
234     // Check if the write_rate is negative
235     else if (write_rate<0){
236         // If write_rate is negative, print an error message and exit the program
237         fprintf(stderr,"Unable to do multithreading use negative rate !\n");
238         printf_info_terminal_by_txt();
239         exit(1);    // Exit the program with an error code of 1
240     }
241     // Check if the sum of the read_rate and write_rate is grater than 100
242     else if (read_rate+write_rate>100){
243         // If the sum is greater than 100, return the complement
244         return complement = read_rate+write_rate;
245     }
246     // Check if the difference between read_rate and write_rate is negative
247     else if (read_rate-write_rate<0){
248         // If the difference is negative, return the complement
249         return complement = read_rate-write_rate;
250     }
251     // If none of the conditions above are met, return 0
252     else {
253         return 0;
254     }
255 }
```

Η συνάρτηση αυτή (Γραμμές 223-255) παίρνει δύο ακέραιες παραμέτρους, `read_rate` και `write_rate`, και υπολογίζει το συμπλήρωμα του αθροίσματος τους. Στη συνέχεια, επιστρέφει αυτήν την τιμή συμπληρώματος.

Πριν από τον υπολογισμό του συμπληρώματος, η συνάρτηση εκτελεί ορισμένους ελέγχους στις παραμέτρους εισόδου. Εάν είτε ο ρυθμός ανάγνωσης είτε ο ρυθμός εγγραφής είναι αρνητικός, η συνάρτηση εκτυπώνει ένα μήνυμα σφάλματος και βγαίνει από το πρόγραμμα. Εάν το άθροισμα των ποσοστών ανάγνωσης και ρυθμού εγγραφής είναι μεγαλύτερο από 100, η συνάρτηση επιστρέφει το συμπλήρωμα αυτού του αθροίσματος. Εάν η διαφορά μεταξύ `read_rate` και `write_rate` είναι αρνητική, η συνάρτηση επιστρέφει το συμπλήρωμα αυτής της διαφοράς. Εάν δεν πληρούνται καμία από αυτές τις προϋποθέσεις, η συνάρτηση επιστρέφει 0.

Η κύρια συνάρτηση(`main`) λαμβάνει ορίσματα γραμμής εντολών και εκτελεί ορισμένες ενέργειες με βάση τις τιμές αυτών των ορισμάτων.

```

258 // This is the main function of the program
259 int main(int argc,char** argv)
260 {
261     // Declare the variables
262     int read_rate;
263     int write_rate ;
264
265     // Check if the first argument is "readwrite"
266     if (strcmp(argv[1],"readwrite")==0){
267         // If there are 7 arguments, take read write rates from the user input
268         if (argc == 7 ){
269             read_rate = atoi(argv[5]); // Take read_rate from user input
270             write_rate = atoi(argv[6]); // Take write_rate from user input
271         }
272         // Otherwise, get read and write rates from user input functions
273         else{
274             read_rate = read_function_rate(); // Take read_rate from user input function
275             write_rate = write_function_rate(); // Take write_rate from user input function
276         }
277     }
278     // Complement read and write rates
279     int value = complement_rate(read_rate,write_rate);
280     if (value>100 || value <0){
281         // If the complement value is greater than 100 or less than 0, set default values
282         read_rate = R_RATE;
283         write_rate = W_RATE;
284         // Print message
285         printf(LINE);
286         printf("Rates Set Default : read_rate = 10 and write_rate = 90");
287         printf(LINE);
288         sleep(1);
289     }
290     // Check for errors in the arguments
291     error_cather(argv[1],argv[2],argv[3],argv[4]);
292     // Declare variables
293     long int count;
294     int id_num;
295     int threads;
296     cost_of_writes = 0.0; // from bench.h
297     cost_of_reads = 0.0; // from bench.h
298
299     // Set size of threads with random generator
300     id_num = random_generator();
301     pthread_t *id_numbers,*id1_numbers,*id2_numbers;
302     id_numbers = (pthread_t *) malloc (id_num*sizeof(int));
303     id1_numbers = (pthread_t *) malloc (id_num*sizeof(int));
304     id2_numbers = (pthread_t *) malloc (id_num*sizeof(int));
305     // Declare data structs and initialize mutexes
306     struct data argument1 , argument2 ,argument3 ;
307     pthread_mutex_init(&total_writes,NULL);
308     pthread_mutex_init(&total_reads,NULL);
309
310     // Seed random number generator
311     srand(time(NULL));

```

Η main ξεκινά δηλώνοντας δύο ακέραιες μεταβλητές `read_rate` και `write_rate`. Στη συνέχεια, ελέγχει εάν το πρώτο όρισμα της γραμμής εντολών είναι "readwrite" χρησιμοποιώντας τη συνάρτηση `strcmp`. Εάν το όρισμα είναι "readwrite", ελέγχει αν υπάρχουν επτά ορίσματα. Εάν υπάρχουν επτά ορίσματα, παίρνει το πέμπτο και το έκτο όρισμα ως `read_rate` και `write_rate`, αντίστοιχα. Εάν δεν υπάρχουν επτά ορίσματα, καλεί δύο συναρτήσεις εισόδου χρήστη για να πάρει τον `read_rate` και τον `write_rate` από τον χρήστη. Μετά τη λήψη

του read_rate και write_rate, ο κώδικας καλεί μια συνάρτηση με το όνομα complement_rate για να υπολογίσει το συμπλήρωμα των read_rate και write_rate. Εάν η τιμή του συμπληρώματος είναι μεγαλύτερη από 100 ή μικρότερη από 0, ο read_rate και write_rate ορίζονται στις προεπιλεγμένες τιμές και εκτυπώνεται ένα μήνυμα. Στη συνέχεια, ο κώδικας καλεί μια συνάρτηση με το όνομα error_cather για να ελέγξει για σφάλματα στα ορίσματα της γραμμής εντολών. Μετά από αυτό, δηλώνει μεταβλητές και αρχικοποιεί ορισμένες μεταβλητές που σχετίζονται με το multi-threading. Τέλος, ο κώδικας δημιουργεί τη γεννήτρια τυχαίων αριθμών.

```

313 // Check if the first argument is "write"
314 if (strcmp(argv[1], "write") == 0) {
315
316     int r = 0;
317     // Get count and threads from user input
318     count = atoi(argv[2]);
319     _print_header(count);
320     _print_environment();
321     if (argc == 5)
322         r = 1;
323     threads = atoi(argv[3]);
324     // Open database and set arguments
325     _open_db();
326     argument1.number_r = r ;
327     argument1.number_count = count ;
328     argument1.number_threads = threads;
329     // Create threads for write requests
330     for (int i = 0 ; i<threads;i++){
331         pthread_create(&id1_numbers[i],NULL,write_request,(void *)&argument1);
332     }
333     // Join threads
334     for (int j = 0 ; j<threads ; j++){
335         pthread_join(id1_numbers[j],NULL);
336     }
337     // Close database and print write time
338     _close_db();
339     print_time_write(argument1.number_count,cost_of_writes);
340
341 }
```

Αυτός ο κώδικας ελέγχει εάν το δεύτερο όρισμα της γραμμής εντολών είναι "write" χρησιμοποιώντας τη συνάρτηση strcmp. Εάν το όρισμα είναι "write", ο κώδικας αρχικοποιεί ορισμένες μεταβλητές που σχετίζονται με το multi-threading και ανοίγει μια βάση δεδομένων. Στη συνέχεια, δημιουργεί έναν καθορισμένο αριθμό νημάτων (καθορίζεται από το τρίτο όρισμα γραμμής εντολών) και καλεί μια συνάρτηση με το όνομα write_request, περνώντας μια δομή με το όνομα argument1 ως όρισμα σε κάθε νήμα. Αφού ολοκληρωθούν όλα τα νήματα, ο κώδικας κλείνει τη βάση δεδομένων και καλεί μια συνάρτηση με το όνομα print_time_write για να εκτυπώσει το χρόνο που χρειάστηκε για την ολοκλήρωση των αιτημάτων εγγραφής. Η δομή με το όνομα argument1 περιέχει τρεις ακέραιες μεταβλητές: number_r, number_count και number_threads, οι οποίες χρησιμοποιούνται για να ορίσουν τα ορίσματα για τη συνάρτηση write_request. Η μεταβλητή count λαμβάνεται από το τρίτο όρισμα γραμμής εντολών και η μεταβλητή threads λαμβάνεται από το τέταρτο όρισμα γραμμής εντολών. Η μεταβλητή number_r ορίζεται σε 0 εάν υπάρχουν τέσσερα ορίσματα γραμμής εντολών και 1 εάν υπάρχουν πέντε ορίσματα γραμμής εντολών. Συνολικά, αυτός ο

κώδικας φαίνεται να υλοποιεί μια λειτουργία εγγραφής βάσης δεδομένων πολλαπλών νημάτων με παραμέτρους που καθορίζονται από τον χρήστη μέσω ορισμάτων γραμμής εντολών.

```
342 // Check if the first argument is "read"
343 else if (strcmp(argv[1], "read") == 0) {
344     // Initialize a variable to hold a flag that will determine whether a read request was performed
345     int r = 0;
346     // Make the second argument as an integer and assign it to the variable "count"
347     count = atoi(argv[2]);
348     // Print a header containing information about the number of requests to be performed
349     _print_header(count);
350     // Print information about the environment
351     _print_environment();
352     //check if there is a fourth argument which indicates whether a read_rate was performed
353     if (argc == 5)
354         r = 1;
355     // Make the third argument as an integer and assign it to the variable "threads"
356     threads = atoi(argv[3]);
357     // Open a database connection
358     _open_db();
359     // Give the struct information about the read request
360     argument1.number_r=r;
361     argument1.number_count =count;
362     argument1.number_threads = threads;
363     // Create "threads" number of threads using "pthread_create", each calling a function named "read_request"
364     for (int i = 0 ; i<threads;i++){
365         pthread_create(&id1_numbers[i],NULL,read_request,(void *)&argument1);
366     }
367     // Wait for all threads to finish using "pthead_join"
368     for (int j = 0 ; j<threads ; j++){
369         pthread_join(id1_numbers[j],NULL);
370     }
371     // Close the database connection
372     _close_db();
373     // Print the time taken to complete the read request
374     print_time_read(argument1.number_count,cost_of_reads);
375 }
```

Αυτός ο κώδικας ελέγχει εάν το δεύτερο όρισμα που μεταβιβάστηκε στο πρόγραμμα είναι "read". Εάν είναι, διαβάζει το τρίτο όρισμα ως τον αριθμό των αιτημάτων που πρέπει να εκτελεστούν (αποθηκευμένα στο πλήθος count) και το τέταρτο όρισμα ως τον αριθμό των νημάτων που θα χρησιμοποιηθούν (αποθηκευμένα στα νήματα της threads). Ελέγχει επίσης εάν υπάρχει ένα πέμπτο όρισμα (argc == 5), το οποίο θα υποδείκνυε ότι καθορίστηκε read_write. Αν ναι, η μεταβλητή r ορίζεται σε 1. Στη συνέχεια, ανοίγει μια σύνδεση βάσης δεδομένων χρησιμοποιώντας τη συνάρτηση _open_db(), αρχικοποιεί τη δομή argument1 με πληροφορίες σχετικά με το αίτημα read και δημιουργεί αριθμό νημάτων χρησιμοποιώντας pthread_create(), καθένα από τα οποία καλεί τη συνάρτηση read_request(). Αφού όλα τα νήματα ολοκληρώσουν την εργασία τους, η σύνδεση της βάσης δεδομένων κλείνει χρησιμοποιώντας τη συνάρτηση _close_db() και ο χρόνος που απαιτείται για την ολοκλήρωση της αίτησης ανάγνωσης εκτυπώνεται χρησιμοποιώντας τη συνάρτηση print_time_read().

```

377 // Check if the first argument is "readwrite"
378 else if(strcmp(argv[1],"readwrite")==0){
379     // Calculate the write and read percentages based on the write_rate
380     double write_rate_1 = (double)write_rate;
381     double read_rate_1 = (double)read_rate;
382     double write_percentage = write_rate_1/100;
383     double read_percentage = read_rate_1/100;
384
385     // Print the write and read percentages to the console
386     printf("\n Write percentage = %f and Read percentage = %f ",write_percentage,read_percentage);
387     // Set r to 1 if argc == 5, otherwise r = 0
388     int r =0 ;
389     count = atoi(argv[2]);
390     _print_header(count);
391     _print_environment();
392     if (argc == 5)
393         r = 1;
394     // Set the number of threads and number of operations for write and read
395     threads = atoi(argv[3]);
396
397     _open_db();
398     // Arguments for the writes
399     argument2.number_count = (long)(count * write_percentage);
400     argument2.number_threads=(int)(threads * write_percentage);
401     argument2.number_r = r;
402     // Arguments for the reads
403     argument3.number_count = (long)(count * read_percentage);
404     argument3.number_threads = (int)(threads * read_percentage);
405     argument3.number_r = r;
406     // Create threads for writing
407     for (int i = 0 ; i<(threads*write_percentage);i++){
408         pthread_create(&id1_numbers[i],NULL,write_request,(void *)&argument2);
409     }
410     // Create threads for reading
411     for (int i = 0 ; i<(threads*read_percentage);i++){
412         pthread_create(&id2_numbers[i],NULL,read_request,(void *)&argument3);
413     }
414     // Wait for reading threads to finish
415     for (int j = 0 ; j<(threads * read_percentage) ; j++){
416         pthread_join(id2_numbers[j],NULL);
417     }
418     // Wait for writing threads to finish
419     for (int j = 0 ; j<(threads*write_percentage) ; j++){
420         pthread_join(id1_numbers[j],NULL);
421     }
422     _close_db();
423     // Print the time it took to perform the writes and reads
424     print_time_write(argument2.number_count,cost_of_writes);
425     print_time_read(argument3.number_count,cost_of_reads);
426 }
```

Αυτός ο κώδικας εκτελεί λειτουργίες read και write σε μια βάση δεδομένων χρησιμοποιώντας πολλαπλά νήματα. Αρχικά ελέγχει εάν το δευτερό όρισμα γραμμής εντολών που διαβιβάστηκε στο πρόγραμμα είναι "readwrite". Εάν είναι, υπολογίζει το ποσοστό των λειτουργιών read και write με βάση το όρισμα write_rate και read_rate που μεταβιβάστηκε στο πρόγραμμα και τις εκτυπώνει στην κονσόλα. Στη συνέχεια, το πρόγραμμα ορίζει τον αριθμό των νημάτων και τον αριθμό των πράξεων για read και write με βάση τα ορίσματα εισόδου που διαβιβάζονται στο πρόγραμμα. Δημιουργεί νήματα για read και write και περιμένει να τελειώσουν τα νήματα χρησιμοποιώντας το pthread_join(). Εκτυπώνει επίσης το χρόνο που χρειάστηκε για την εκτέλεση των read και των write.

```
427     else {
428         // Print usage instructions if the command line arguments are incorrect
429         fprintf(stderr,"Usage: db-bench <write | read | readwrite> <count> <threads> <r> \n");
430         exit(1);
431     }
432     // Return 1 to indicate successful completion
433     return 1;
434 }
```

Εάν το δεύτερο όρισμα γραμμής εντολών που μεταβιβάστηκε στο πρόγραμμα δεν είναι "write", "read" ή "readwrite", εκτυπώνει οδηγίες χρήσης στην κονσόλα και εξέρχεται από το πρόγραμμα με κωδικό εξόδου 1, υποδεικνύοντας ένα σφάλμα. Οι οδηγίες χρήσης καθορίζουν τη σωστή μορφή για τα ορίσματα της γραμμής εντολών, συμπεριλαμβανομένης της λειτουργίας προς εκτέλεση, τον αριθμό των λειτουργιών που πρέπει να εκτελεστούν (μέτρηση), τον αριθμό των νημάτων που θα χρησιμοποιηθούν και ένα προαιρετικό "r" διαφωνία. Εάν τα ορίσματα της γραμμής εντολών είναι σωστά και το πρόγραμμα ολοκληρωθεί με επιτυχία, επιστρέφει μια τιμή 1.

BENCH.H ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΑΛΛΑΓΩΝ – ΤΡΟΠΟΠΟΙΗΣΕΩΝ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <stdint.h>
5 #include <string.h>
6 #include <time.h>
7 #include <sys/time.h>
8 // include the pthread library
9
10 #define KSIZE (16)
11 #define VSIZE (1000)
12
13 #define LINE "+-----+\n"
14 #define LINE1 "------\n"
15
16 long long get_ustime_sec(void);
17 void _random_key(char *key,int length);
18
19 pthread_mutex_t total_writes; // initialize a mutex for total writes
20 pthread_mutex_t total_reads; // initialize a mutex for total reads
21 double cost_of_writes; // initialize the cost of writes
22 double cost_of_reads; // initialize the cost of reads
23
24 struct data // define a struct to hold some data
25 {
26     int number_r; // random number r
27     long int number_count; // total count
28     int number_threads; // number of threads
29 };
```

Γραμμή 8: Το αρχείο κεφαλίδας pthread.h περιέχει δηλώσεις συναρτήσεων και αντιστοιχίσεις για διασυνδέσεις νήματος και ορίζει έναν αριθμό σταθερών που χρησιμοποιούνται από αυτές τις συναρτήσεις.

Γραμμές 19-22: Ορίζονται δύο mutexes χρησιμοποιώντας το pthread_mutex_t: total_writes και total_reads. Αυτά χρησιμοποιούνται για την πρόσβαση σε κοινόχρηστες μεταβλητές του προγράμματος. Ορίζονται επίσης δύο δεκαδικές μεταβλητές cost_of_writes και cost_of_reads, οι οποίες χρησιμοποιούνται για την αποθήκευση του κόστους εγγραφής και ανάγνωσης, αντίστοιχα.

Γραμμές 24-28: Ορίζεται μια δομή με όνομα data με τρία μέλη: number_r, number_count και number_threads. Αυτή η δομή χρησιμοποιείται για τη διατήρηση ορισμένων δεδομένων που σχετίζονται με το multi-threading στο πρόγραμμα.

KIWI.C ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΑΛΛΑΓΩΝ - ΤΡΟΠΟΠΟΙΗΣΕΩΝ

Κυρίως σκοπος της kiwi.c

Η kiwi έχει ως στόχο

- Την επικοινωνία με την βάση δεδομένων για να γίνονται οι λειτουργίες db_add() και db_get()
ανάλογα με το πόσα threads και των αριθμό λειτουργιών έχει ζητήσει ο χρήστης
- Την προβολή αποτελεσμάτων χρόνου.

Περιγραφή και φωτογραφίες read_test()

```
86 void _read_test(long int count, int r, int threads){  
87  
88     int i;  
89     long int counter;  
90     int ret;  
91     int found = 0;  
92     double cost;  
93     long long start,end;  
94     Variant sk;  
95     Variant sv;  
96     char key[KSIZE + 1];  
97  
98     start = get_ustime_sec();  
99     // Call the 'find_rate' function and assign the result to the 'counter' variable  
100    counter = find_rate(count, threads);  
101    for (i = 0; i < counter; i++) {  
102        memset(key, 0, KSIZE + 1);  
103  
104        if (r)  
105            _random_key(key, KSIZE);  
106        else  
107            sprintf(key, KSIZE, "key-%d", i);  
108        fprintf(stderr, "%d searching %s\n", i, key);  
109        sk.length = KSIZE;  
110        sk.mem = key;  
111  
112        ret = db_get(db, &sk, &sv);  
113        if (ret) {  
114            found++;  
115        } else {  
116            INFO("not found key#%s",  
117                  sk.mem);  
118        }  
119  
120        if ((i % 10000) == 0) {  
121            fprintf(stderr,"random read finished %d ops%30s\n", i, "");  
122            fflush(stderr);  
123        }  
124  
125        end = get_ustime_sec();  
126        cost = end - start;  
127        // Lock the mutex associated with the 'total_reads' variable to prevent access  
128        pthread_mutex_lock(&total_reads);  
129        // Add the cost of the reads to the 'cost_of_reads' variable  
130        cost_of_reads = cost_of_reads + cost;  
131        // Unlock the mutex associated with the 'total_reads' variable to allow other threads to access it  
132        pthread_mutex_unlock(&total_reads);  
133    }  
134 }
```

Γραμμή (86): Ορισμός συνάρτησης **void _read_test(long int count, int r, int threads)**

- Ορίσματα: Τα ορίσματα έχουν οριστεί στην `bench.c` και τα έχουμε περάσει σε ένα struct στην `bench.h`
 - **long int count** είναι το πόσες λειτουργίες στο σύνολο θα κάνουμε
 - **int r**
 - **int threads** είναι το πόσα νήματα θα χρησιμοποιήσουμε στην
- Return Value τύπου void

Γραμμή (98):Στην αρχή παίρνουμε τον χρόνο τον και τον βάζουμε σε μια μεταβλητή start ώστε να ξέρουμε πότε ξεκίνησε το test

Γραμμή (99):Βρίσκουμε τον λόγο των συνολικών διεργασιών σε σχέση με τα threads με την χρήση της `find_rate(count, threads)` για να ξέρουμε πόσες διεργασίες θα εκτελέσουμε μέσα στην βάση δεδομένων

Γραμμή (101-125): μπαίνουμε μέσα σε ένα for loop στο οποίο εκτελούμε την `db_get()` και τερματίζει μέχρι να γίνει μια temp μεταβλητή i ίση με το λόγο (Αρκετά είναι ίδιο με το αρχικό `kiwi.c`).

Γραμμή (126) παίρνουμε την δεδομένη ώρα σε sec από τον υπολογιστή ώστε να υπολογίσουμε μετέπειτα πόσο χρειάστηκε το read test που εξομοιώσαμε. Εκτελούμε μια αφαίρεση ώστε να δούμε πόσο χρόνο χρειάστηκε .

Γραμμή (128 – 133):Στις γραμμές εκτελούμε αμοιβαίο αποκλεισμό ώστε να βρούμε το συνολικό χρόνο όλων αυτών των test που γίνονται .

Περιγραφή και φωτογραφίες `write_test()`

```
32 void _write_test(long int count, int r, int threads){  
33  
34     int i;  
35     long int counter;  
36     double cost;  
37     long long start,end;  
38     Variant sk, sv;  
39  
40     char key[KSIZE + 1];  
41     char val[VSIZE + 1];  
42     char sbuf[1024];  
43  
44     memset(key, 0, KSIZE + 1);  
45     memset(val, 0, VSIZE + 1);  
46     memset(sbuf, 0, 1024);  
47  
48     start = get_ustime_sec();  
49  
50     counter = find_rate(count, threads);  
51     for (i = 0; i < counter; i++) {  
52         if (r)  
53             _random_key(key, KSIZE);  
54         else  
55             sprintf(key, KSIZE, "key-%d", i);  
56         fprintf(stderr, "%d adding %s\n", i, key);  
57         sprintf(val, VSIZE, "val-%d", i);  
58  
59         sk.length = KSIZE;  
60         sk.mem = key;  
61         sv.length = VSIZE;  
62         sv.mem = val;  
63  
64         if ((i % 10000) == 0)  
65             fprintf(stderr,"random write finished %d ops%30s\r", i, "");  
66             fflush(stderr);  
67     }  
68 }  
69  
70 end = get_ustime_sec();  
71 cost = end - start; // the time in seconds that a operation needs to be done  
72  
73 // Lock the mutex associated with the 'total_writes' variable to prevent access  
74 pthread_mutex_lock(&total_writes);  
75  
76 // Add the cost of the writes to the 'cost_of_writes' variable  
77 cost_of_writes = cost_of_writes + cost;  
78  
79 // Unlock the mutex associated with the 'total_writes' variable to allow other threads to access it  
80 pthread_mutex_unlock(&total_writes);  
81 }  
82 }
```

Γραμμή (32): Ορισμός συνάρτησης `void _write_test(long int count, int r, int threads)`

- Ορίσματα: Τα ορίσματα έχουν οριστεί στην `bench.c` και τα έχουμε περάσει σε ένα `struct` στην `bench.h`
 - **long int count** είναι το πόσες λειτουργίες στο σύνολο θα κάνουμε
 - **int r**
 - **int threads** είναι το πόσα νήματα θα χρησιμοποιήσουμε στην
- Return Value τύπου `void`

Γραμμή (48): Στην αρχή παίρνουμε τον χρόνο που έχει εκείνη την χρονική στιγμή που ξεκινά ένα πείραμα τον και τον βάζουμε σε μια μεταβλητή start ώστε να ξέρουμε πότε ξεκίνησε το test

Γραμμή (50): βρίσκουμε τον λόγο των συνολικών διεργασιών σε σχέση με τα threads με την χρήση της `find_rate(count, threads)` για να ξέρουμε πόσες διεργασίες θα εκτελέσουμε μέσα στην βάση δεδομένων

Γραμμή (51-67): μπαίνουμε μέσα σε ένα for loop στο οποίο κάνουμε εκτελούμε την `db_add()` και τερματίζει μέχρι να γίνει μια temp μεταβλητή i ίση με το λόγο (Αρκετά είναι ίδιο με το αρχικό kiwi.c).

Γραμμή (70-71) παίρνουμε την δεδομένη ώρα σε sec από τον υπολογιστή ώστε να υπολογίσουμε μετέπειτα πόσο χρειάστηκε το read test που εξομοιώσαμε. Εκτελούμε μια αφαίρεση ώστε να δούμε πόσο χρόνο χρειάστηκε .

Γραμμή (73-81): Στις γραμμές εκτελούμε αμοιβαίο αποκλεισμό ώστε να βρούμε το συνολικό χρόνο όλων αυτών των test που γίνονται .

Περιγραφή και φωτογραφίες find_rate()

```
23 // Function to find the rate for distributing data across threads
24 long int find_rate(long int count, int threads){
25     long int counter;
26     // Calculate the number of data elements to assign to each thread
27     counter = count/thread;
28     return counter; // Return the calculated value
29 }
```

Μια απλή συνάρτηση η οποίο επιστρέφει έναν integer ο οποίος μας βοηθάει να βρούμε πόσες φορές θα εκτελεσθεί μια function μέσα στην βάση δεδομένων.

DB.H ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΑΛΛΑΓΩΝ – ΤΡΟΠΟΠΟΙΗΣΕΩΝ

```
10  typedef struct _db {  
11      //    char basedir[MAX_FILENAME];  
12  
13      pthread_mutex_t readwrite_mutex;  
14      pthread_mutex_t readcount_mutex;  
15      int readcount;  
16      char basedir[MAX_FILENAME+1];  
17      SST* sst;  
18      MemTable* memtable;  
19 } DB;  
20
```

Μέσα στο μια δομή(struct) η οποία προϋπήρχε βάζουμε κάποιες μεταβλητές τύπου pthread_mutex_t οι οποίες θα χρησιμοποιηθούν μετέπειτα μέσα στην db.c που την περνάμε σαν όρισμα στην συνάρτηση db_get(DB* self, Variant* key, Variant* value) και στην db_add(DB* self, Variant* key, Variant* value).

1.Η μεταβλητή **pthread_mutex_t** readwrite_mutex χρησιμοποιείτε κυρίως για τον ταυτοχρονισμό του write και πρέπει να είναι καθολική μεταβλητή για αυτό την ορίζουμε στην db.h και περνιέται σαν όρισμα μέσα στις pthread_mutex_lock()και pthread_mutex_unlock()

2.Η μεταβλητή **pthread_mutex_t** readcount_mutex χρησιμοποιείται για τον ταυτοχρονισμό του read και να είναι και αυτή καθολική μεταβλητή για αυτό την ορίζουμε στην db.h αι περνιέται σαν όρισμα μέσα στις pthread_mutex_lock()και pthread_mutex_unlock()

3. int readcount o counter αυτός θα χρησιμοποιηθεί μέσα στην **db_get()** μετράει το πόσους εγγραφές/writers θα έχουμε για να αποκλείουμε κάποιες περιπτώσεις οι οποίες θα μας οδηγήσουν σε λάθος ταυτοχρονισμό μέσα στη βάση δεδομένων μεταξύ των πολλαπλών readers που θέλουμε να έχουμε αυτό φαίνεται καλυτέρα παρακάτω

(Θα μπορούσαμε να χρησιμοποιήσουμε επίσης global τις μεταβλητές τύπου pthread_mutex_t μέσα στην db.c θεωρήσαμε ότι είναι καλύτερο να είναι σε ένα header file σε μια δομή η οποία θα περνάει και σαν όρισμα σε μια συνάρτηση)

DB.C ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΑΛΛΑΓΩΝ – ΤΡΟΠΟΠΟΙΗΣΕΩΝ

Ερώτημα – (iv) : Προσδιορίστε το επίπεδο ταυτοχρονισμού που θέλετε να πετύχετε. Αυτό εξαρτάται από τα μέρη της δομής που εμπλέκονται στις διάφορες λειτουργίες και το αν διαβάζουν μόνο τα δεδομένα ή επιπλέον τροποποιούν τα δεδομένα. Μια απλή προσέγγιση είναι να επιτρέψετε να τρέχει μία μόνο λειτουργία τη φορά και να το εξασφαλίσετε αυτό με αμοιβαίο αποκλεισμό. Θα πρέπει να βελτιώσετε αυτή την τετριμένη λύση προσθέτοντας επιπλέον ταυτοχρονισμό με την χρήση πολλαπλών κρίσιμων περιοχών και πειραματισμό με συγχρονισμό reader-writer στα διάφορα επίπεδα του LSM-tree (π.χ., skip list, sst, συγχώνευση, σύμπτυξη, κλπ.).

Κυρίως σκοπός της db.c

Κυρίως σκοπός γενικά του engine package και του αρχείου db.c είναι να εξυπηρετήσουν το backend . Υπεύθυνο για τα αιτήματα των χρηστών και το χειρισμό δεδομένων και την αποστολή απαντήσεων πίσω στην πλευρά

Λειτουργίες της βάσης δεδομένων είναι

db_get()

Ως κύρια λειτουργία έχει να διαβάζει από

1. <μνήμη Ram = memtable_get()>
2. είτε άμα υπάρξει το compaction διαβάζει από το <σκληρό δίσκο = sst_get()>

db_add()

Ως κύρια λειτουργία έχει να γραφει σε

1. <μνημη Ram = memtable_get()>
2. είτε άμα υπάρξει το compaction γράψει στο το <σκληρό δίσκο = sst_get()>

Περιγραφή Προβλήματος

Αρχικά έχουμε προσέγγιση τις δυο βασικές λειτουργίες της db_add() και της db_get() χρησιμοποιώντας την μέθοδο του αμοιβαίου αποκλεισμού δηλαδή εκτελείται μια λειτουργία την φορά επομένως είτε αφήνει το γράψιμο είτε διάβασμα από την βάση δεδομένων .

Λύση Προβλήματος

Βελτιωμένη προσέγγιση των αιτημάτων στην βάση δεδομένων μπορεί να γίνει με την χρήση ταυτοχρονισμού με την χρήση πολλαπλών κρίσιμων περιοχών και πειραματισμό με συγχρονισμό reader-writer στα διάφορα επίπεδα του LSM-tree.

Συγχρονισμός reader-writer πολλαπλών κρίσιμων περιοχών

1. Ένα σύνολο δεδομένων μοιράζεται σε έναν αριθμό διαδικασιών
2. Όταν μια εγγραφή είναι έτοιμη τότε ολοκληρώνεται η εγγραφή στην βάση δεδομένων (μόνο ένας μπορεί να γραφει την φορά)
3. Όταν εκτελείται μια εγγραφή , δεν μπορεί να εκτελείται γράψιμο
4. Όταν εκτελείται ανάγνωση από την βάση δεδομένων , δεν μπορεί να γίνει κανένα γράψιμο στην βάση δεδομένων
5. Ένας αναγνώστης μπορεί μόνο να διαβάσει

Υπόθεση	Διαδικασία 1	Διαδικασία 2	Επιτρέπεται
Περίπτωση 1	<u>db_add()</u>	<u>db_add()</u>	False
Περίπτωση 2	<u>db_add()</u>	<u>db_get()</u>	False
Περίπτωση 3	<u>db_get()</u>	<u>db_add()</u>	False
Περίπτωση 4	<u>db_get()</u>	<u>db_get()</u>	True

Αυτό σημαίνει ότι όταν πραγματοποιούμε **db_get()** δεν χρειάζεται να περιμένουμε μέχρι να τελειώσει μια ανάγνωση για να εκτελέσουμε ταυτόχρονα ακόμα μια ή και πολλές μαζί .

3 **#include "db.h"**

χρειάζεται ώστε να έχουμε πρόσβαση στις μεταβλητές, δομές, συναρτήσεις που έχουμε ορίσει στην db.h

Περιγραφή και φωτογραφίες συνάρτησης db_get()

```
78  /*function manage read in database
79  *with read-write synchronization
80  *reader enter critical area to read in database
81  *increase by one a counter which count the readers
82  *if a reader is the first and a writer comes need to stop it and not accept to write simultaneously in database
83  *we need to give permission to a lot of readers in database
84  *!returns an integer with the read value
85 */
86 int db_get(DB* self, Variant* key, Variant* value)
87 {
88     int read_value;
89
90
91     //reader enter in critical area
92     pthread_mutex_lock(&self->readcount_mutex);
93     //with this counter increase exclude the situation if a writer enter in critical area
94     self->readcount++;
95     //there is at least one reader //
96     if (self->readcount == 1){
97         pthread_mutex_lock(&self->readwrite_mutex);
98     }
99     // leave of critical area to allow others readers to run simultaneously
100    pthread_mutex_unlock(&self->readcount_mutex);
101    // check if need to search in memtable or in sst to find the value
102    if (memtable_get(self->memtable->list, key, value) == 1){
103        read_value = 1;
104    }
105    else{
106        read_value = sst_get(self->sst, key, value);
107    }
108 }
```

```
110     pthread_mutex_lock(&self->readcount_mutex);
111     self->readcount--;
112     if(self->readcount==0)//if counter=0 exit because there is no one to read
113         pthread_mutex_unlock(&self->readwrite_mutex);//allow writers to write(db_add())
114     // exit critical area
115     pthread_mutex_unlock(&self->readcount_mutex);
116     return read_value;
117 }
118
119 int db_remove(DB* self, Variant* key)
120 {
121     return memtable_remove(self->memtable, key);
122 }
```

Δεν θα σχολιάσουμε τα comments που έχουμε βάλει για να εξηγήσουμε των κώδικα στην db.c

Γραμμή (79 - 85): σχόλια που περιγράφουν γενικά πως λειτουργεί η συνάρτηση

Γραμμή (86): Ορισμός συνάρτησης int db_get(DB* self, Variant* key, Variant* value)

- Ορίσματα
 - DB self ώστε να μπορεί να έχει τις μεταβλητές από το struct DB που είναι στην db.h
 - Variant Key παίρνει σαν ορίσματά κλειδί που ψάχνει μέσα στην βάση δεδομένων
 - Variant Value παίρνει σαν όρισμα το value που ψάχνει να βρει μέσα στην βάση
- Return Value τύπου Int από πριν έχει οριστεί τι θα επιστρέψει χωρίς κάποια αλλαγή

Γενικά με την μεθοδολογία που θα εφαρμόσουμε ταυτοχρονισμού των λειτουργιών παρακάτω και με την εξήγηση που θα δώσουμε θα πρέπει να αποκλείσουμε την περίπτωση να έχουμε μαζί κάποιων reader και writer στην βάση δεδομένων . Αλλά θα πρέπει να επιτρέπουμε την ταυτόχρονη λειτουργία πολλαπλών readers. Αυτό θα δείξουμε παρακάτω .

Γραμμή (89): είναι η μεταβλητή επιστροφής τύπου int που θα επιστρέφει η συνάρτηση όταν γίνει κάποιο return μέσα της

Γραμμή (92-99) : ΕΙΣΑΓΩΓΗ ΚΡΙΣΙΜΗ ΠΕΡΙΟΧΗ ΕΝΟΣ READER ΚΑΙ ΑΠΟΚΛΕΙΣΜΟΣ WRITER

Μπαίνουμε αρχικά στην κρίσιμη περιοχή χρησιμοποιώντας κλειδαριές για να κάνουμε read από την βάση δεδομένων . Αρχικά , κλειδώνουμε με την συνάρτηση **pthread_mutex_lock(&self->readcount_mutex)** και αυξάνουμε την μεταβλητή readcount η οποία βοηθάει στο να αποκλείσουμε το ενδεχομένο να μπει κάποιος writer στη κρίσιμη περιοχή και κλειδώνουμε τον writer με την **pthread_mutex_lock(&self->readwrite_mutex)** δεν θέλουμε στον ταυτοχρονισμό που θα χρησιμοποιήσουμε όπως είπαμε να γίνεται ταυτόχρονα read και write.Μετέπειτα, ξεκλειδώνουμε για να μπορέσουμε να βάλουμε πολλαπλούς reader .

Γραμμή (103-109) : Φορτώνουμε την τιμή από το memtable ή από το sst στην read_value

Γραμμή (110-116) : Εκτελούμε πολλαπλά read από την βάση

Μπαίνουμε στην κρίσιμη περιοχή για να εκτελέσουμε πολλαπλά κλειδώνοντας με την **pthread_mutex_lock(&self->readcount_mutex)** για να κάνουμε read ανάλογα πόσα έχουμε και αφαιρούμε κάθε φορά ένα από τον read count μέχρι να μηδενιστεί για να αφήσουμε ελεύθερα να κάνει write από την βάση αφού έχουν τελειώσει πλέον οι readers και έχει γίνει ο counter = 0 ξεκλειδώνουμε ώστε να βγούμε από την κρίσιμη περιοχή και επιστρέφουμε την τιμή που κάναμε read από το memtable ή το sst.

Περιγραφή και φωτογραφίες συνάρτησης db_add()

```
51  /*
52   *function which manage writes in database
53   *with read-write synchronization
54   *we allow in this situation only one writer to enter into critical area and leave
55   *!return an integer write value
56   */
57  int db_add(DB* self, Variant* key, Variant* value){
58      int write_value;
59
60      //lock the critical area for write
61      pthread_mutex_lock(&self->readwrite_mutex);
62      //check for compactation this code is from previously here
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
71      //add the key and value in database
72      write_value = memtable_add(self->memtable, key, value);
73      //leaves the critical area
74      pthread_mutex_unlock(&self->readwrite_mutex);
75      return write_value;
76 }
```

Γραμμή (57): Ορισμός συνάρτησης int db_add(DB* self, Variant* key, Variant* value)

- Ορίσματα
 - DB self ώστε να μπορεί να έχει τις μεταβλητές από το struct DB που είναι στην db.h
 - Variant Key παίρνει σαν ορίσματά κλειδί που φτιάχνουμε μέσα στην βάση δεδομένων
 - Variant Value παίρνει σαν όρισμα το value που έχουμε βρει μέσα στην βάση
- Return Value τύπου Int από πριν έχει οριστεί τι θα επιστρέψει χωρίς κάποια αλλαγή από το αρχικό

Γραμμή (58): φτιάχνουμε μια μεταβλητή int οποία θα είναι αυτή που θα επιστρέψει

Γραμμή (61-75):Κάνουμε Write μέσα στην βάση δεδομένων

Στην βάση δεδομένων δεν μπορώ να έχω ταυτόχρονα πολλές εγγραφές ούτε να έχω write και μετά να έχω ταυτόχρονα read αυτό φαίνεται και στον πίνακα παραπάνω. Οπότε στο σημείο του κώδικα είναι πιο απλό από την db_get() αφού χρειάζεται μόλις γίνεται μια εγγραφή να κλειδώνω και μετά να ξεκλειδώνω για να μην έχω πρόβλημα του ταυτοχρονισμού σύμφωνα και με την μέθοδο writer-reader που μας ζητείται να εφαρμόσουμε. Αρχικά, μπαίνουμε στην κρίσιμη περιοχή κλειδώνουμε με την pthread_mutex_lock(&self->readwrite_mutex) και ελέγχουμε εάν χρειάζεται να γίνει κάποιο compactation του memtable(υπάρχει ήδη

αυτό το κομμάτι από πριν) . Έπειτα, βάζουμε στο write_value την memtable_add(self->memtable, key, value) δηλαδή την μεταβλητή που γράφουμε στην βάση δεδομένων και βγαίνουμε από την κρίσιμη περιοχή . Τέλος επιστρέφουμε την μεταβλητή που βάζουμε στην βάση δεδομένων.

Test commands in terminal

./kiwi-bench <read|write|readwrite> <number of processes> <number of threads> <r> <percentage of read> <percentage of read>

1.argv[1] <read|write|readwrite> : εντολή που θέλουμε να εκτελεστεί το πρόγραμμα. Υπάρχουν συναρτήσεις που προλαμβάνουν τα λάθη του χρήστη π.χ. εάν γράψεις λάθος κάποιο από τα 3 είτε εσκεμμένα είτε εν αγνοία σου τότε να σου εμφανίζεται μήνυμα λάθους στην κονσόλα σου και ένα αρχείο <Information.txt> να σου υποδεικνύει το σωστό συντακτικό εντολής που πρέπει να ακολουθήσεις.

2.argv[2] <number of processes> : πόσα processes από read ή write ή readwrite θέλουμε να γίνουν . Υπάρχουν συναρτήσεις που προλαμβάνουν τα λάθη του χρήστη π.χ. εάν έχει βάλει αρνητικό αριθμό είτε εσκεμμένα είτε εν αγνοία σου τότε να σου εμφανίζεται μήνυμα λάθους στην κονσόλα σου και ένα αρχείο <Information.txt> να σου υποδεικνύει το σωστό συντακτικό εντολής που πρέπει να ακολουθήσεις.

3. argv[3] <number of threads> : πόσα νήματα θέλω να χρησιμοποιήσω για την προσομοίωση . Υπάρχουν συναρτήσεις που προλαμβάνουν τα λάθη του χρήστη π.χ. εάν έχει βάλει αρνητικό αριθμό είτε εσκεμμένα είτε εν αγνοία σου τότε να σου εμφανίζεται μήνυμα λάθους στην κονσόλα σου και ένα αρχείο <Information.txt> να σου υποδεικνύει το σωστό συντακτικό εντολής που πρέπει να ακολουθήσεις.

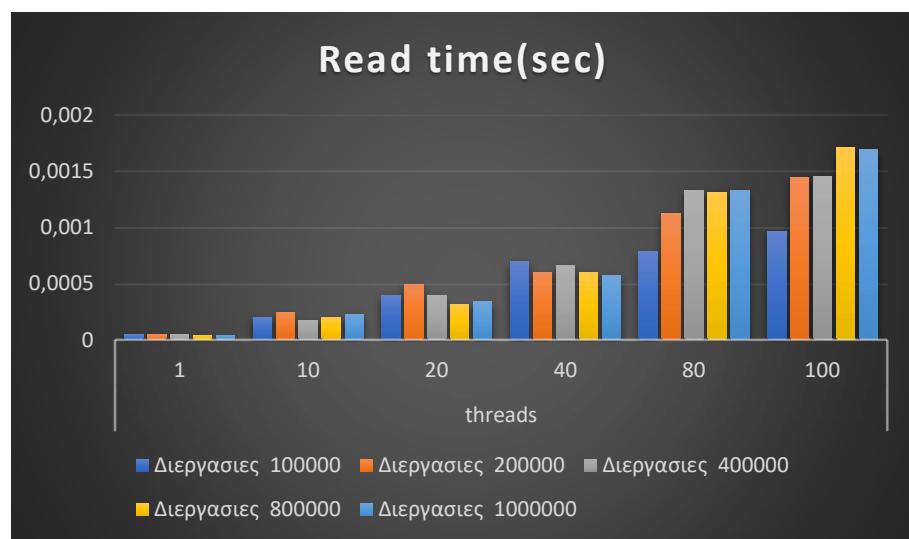
4. argv[4] <r> :

4. argv[5] < percentage of read > : το ποσοστό που θέλω να κάνει read . Υπάρχουν συναρτήσεις που προλαμβάνουν τα λάθη του χρήστη π.χ. εάν έχει βάλει αρνητικό αριθμό είτε εάν έχει βάλει μεγάλο ποσοστό πάνω από 100% τότε τρέχει το default ποσοστό το read και του write μέσα στο υπόλοιπο πρόγραμμα είτε εσκεμμένα είτε εν αγνοία σου τότε να σου εμφανίζεται μήνυμα λάθους στην κονσόλα σου και ένα αρχείο <Information.txt> να σου υποδεικνύει το σωστό συντακτικό εντολής που πρέπει να ακολουθήσεις. Επίσης , όταν έχουμε write ή read μόνο τότε το ποσοστό αυτόματα είναι 100%.

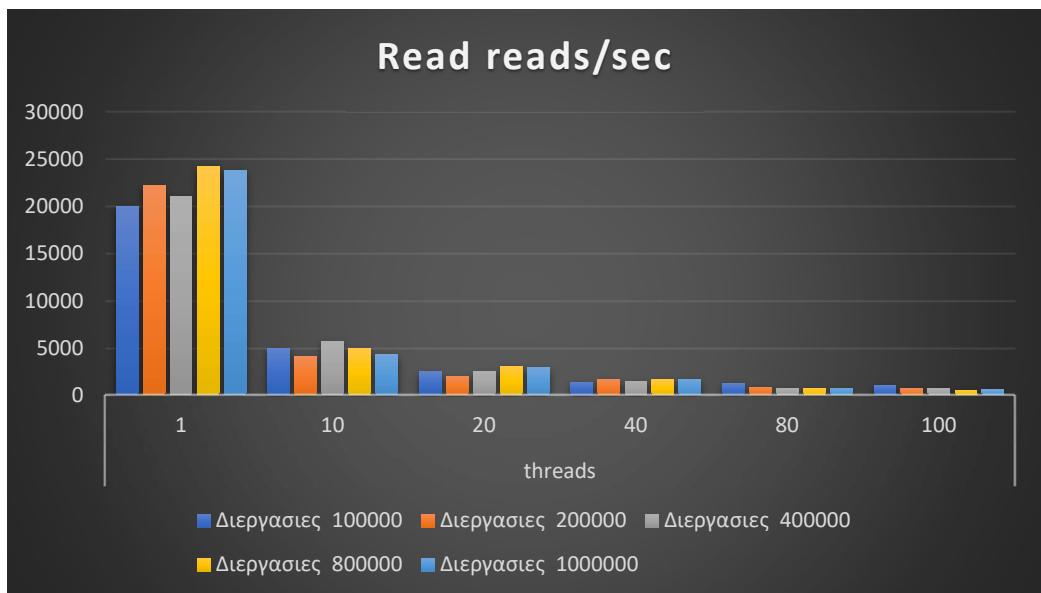
5. argv[6] < percentage of read > : το ποσοστό που θέλω να κάνει write . Υπάρχουν συναρτήσεις που προλαμβάνουν τα λάθη του χρήστη π.χ. εάν έχει βάλει αρνητικό αριθμό είτε εάν έχει βάλλει μεγάλο ποσοστό πάνω από 100% τότε τρέχει το default ποσοστό το read και του write μέσα στο υπόλοιπο πρόγραμμα είτε εσκεμμένα είτε εν αγνοία σου τότε να σου εμφανίζεται μήνυμα λάθους στην κονσόλα σου και ένα αρχείο <Information.txt> να σου υποδεικνύει το σωστό συντακτικό εντολής που πρέπει να ακολουθήσεις. Επίσης , όταν έχουμε write ή read μόνο τότε το ποσοστό αυτόματα είναι 100%.

TEST FOR THE READ

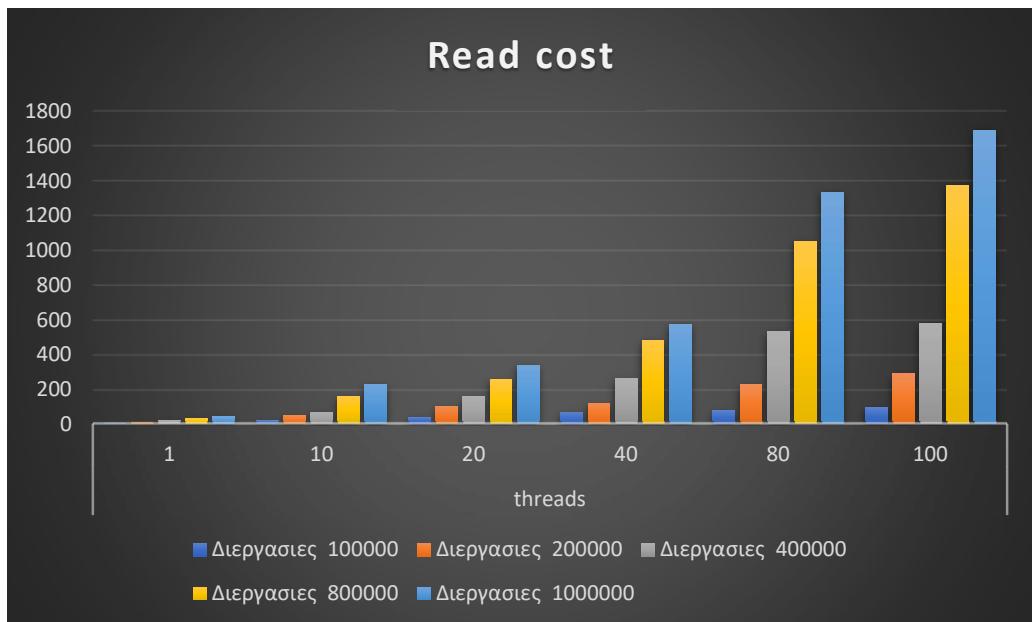
Read time(sec)		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads	1	0.00005	0.000045	0.000048	0.000041	0.000042
	10	0.0002	0.000245	0.000175	0.0002	0.00023
	20	0.0004	0.0005	0.000398	0.000321	0.000339
	40	0.0007	0.0006	0.000663	0.000606	0.000576
	80	0.00079	0.00113	0.001335	0.001318	0.001334
	100	0.00097	0.00145	0.001455	0.001718	0.001693



Read reads/sec		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	1	20000	22222.2	21052.6	24242.4	23809.5
	10	5000	4081.6	5714.3	5000	4347.8
	20	2500	2000	2515.7	3112.8	2949.9
	40	1428.6	1666.7	1509.4	1649.5	1736.1
	80	1265.8	885	749.1	759	749.6
	100	1030.9	689.7	687.3	582.2	590.7

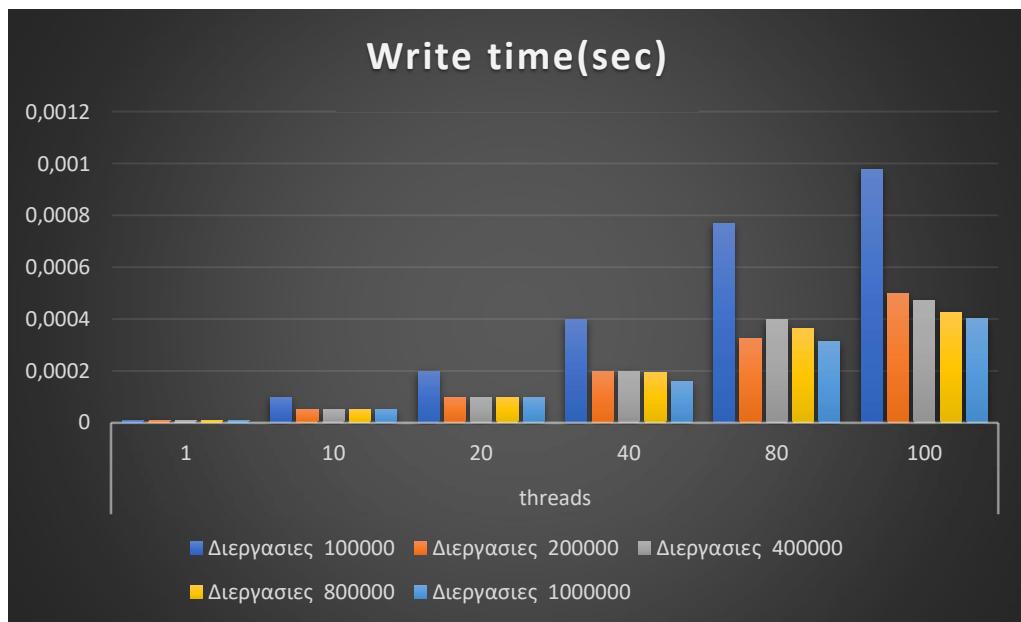


Read cost		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	1	5	9	19	33	42
	10	20	49	70	160	230
	20	40	100	159	257	339
	40	70	120	265	485	576
	80	79	226	534	1054	1334
	100	97	290	582	1374	1693

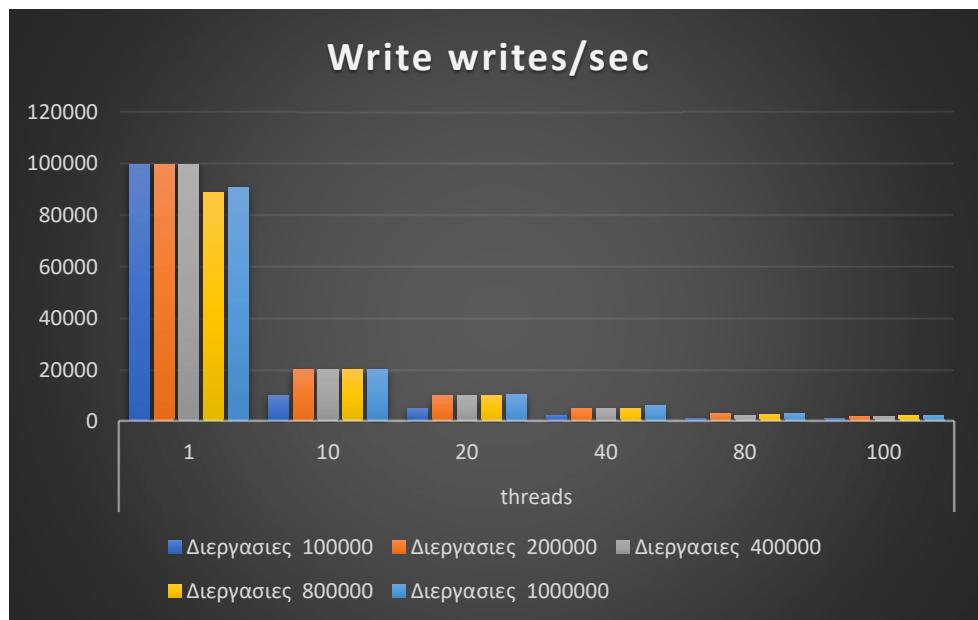


TEST FOR THE WRITE

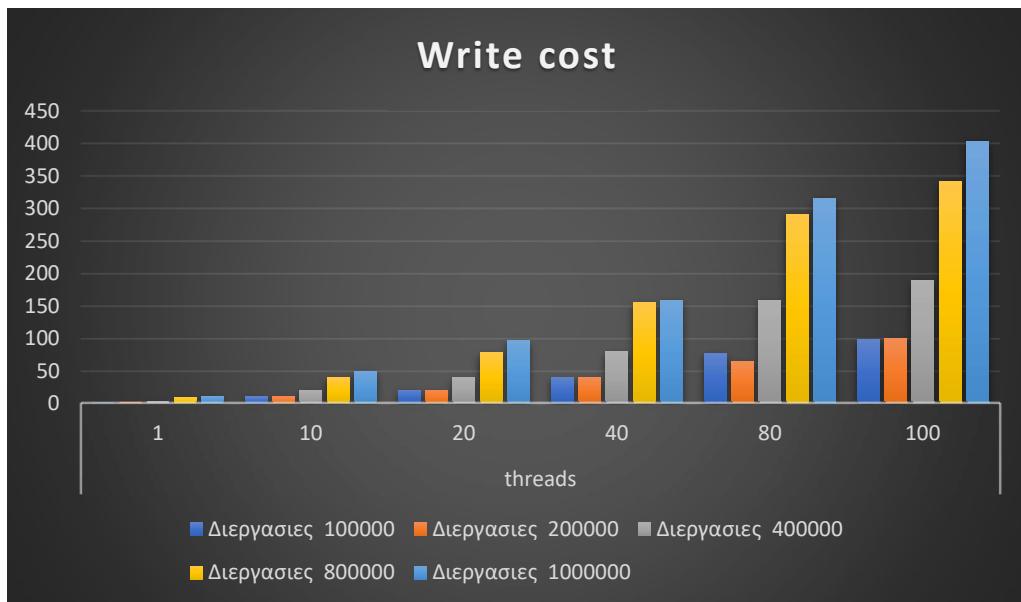
Write time(sec)		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads	1	0.00001	0.00001	0.00001	0.000011	0.000011
	10	0.0001	0.00005	0.00005	0.00005	0.00005
	20	0.0002	0.0001	0.0001	0.000099	0.000097
	40	0.0004	0.0002	0.0002	0.000194	0.000159
	80	0.00077	0.000325	0.000398	0.000364	0.000316
	100	0.00098	0.0005	0.000472	0.000427	0.000404



Write writes/sec		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads	1	100000	100000	100000	88888.9	90909.1
	10	10000	20000	20000	20000	20000
	20	5000	10000	10000	10126.6	10309.3
	40	2500	5000	5000	5161.3	6289.3
	80	1298.7	3076.9	2515.7	2749.1	3164.6
	100	1020.4	2000	2116.4	2339.2	2475.2

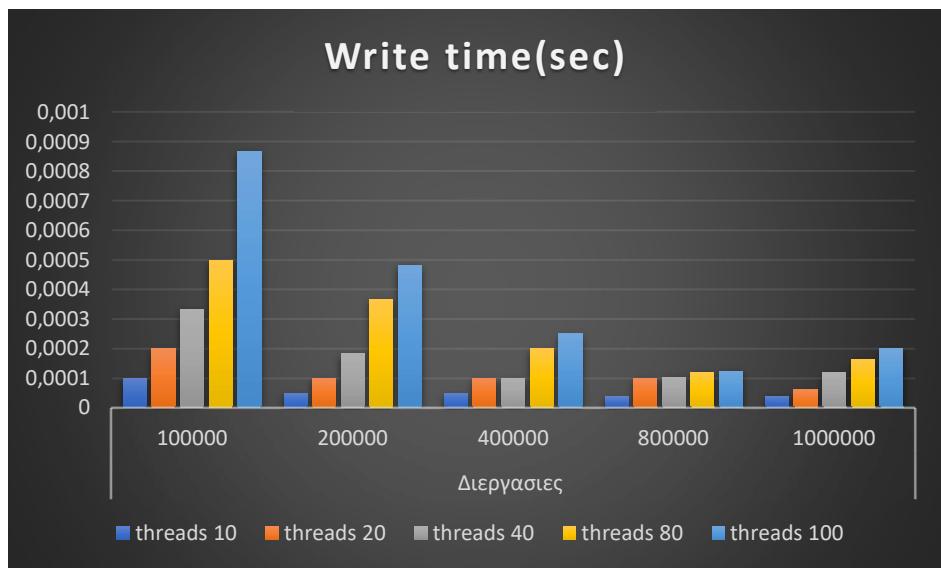


Write cost		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	1	1	2	4	9	11
	10	10	10	20	40	50
	20	20	20	40	79	97
	40	40	40	80	155	159
	80	77	65	159	291	316
	100	98	100	189	342	404

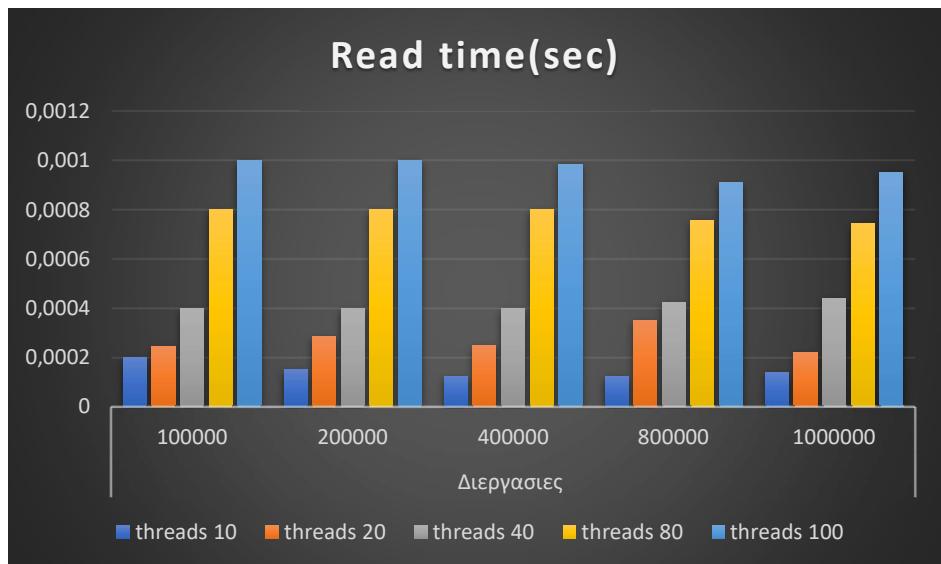


TEST FOR THE READWRITE ME 70% Read 30% Write

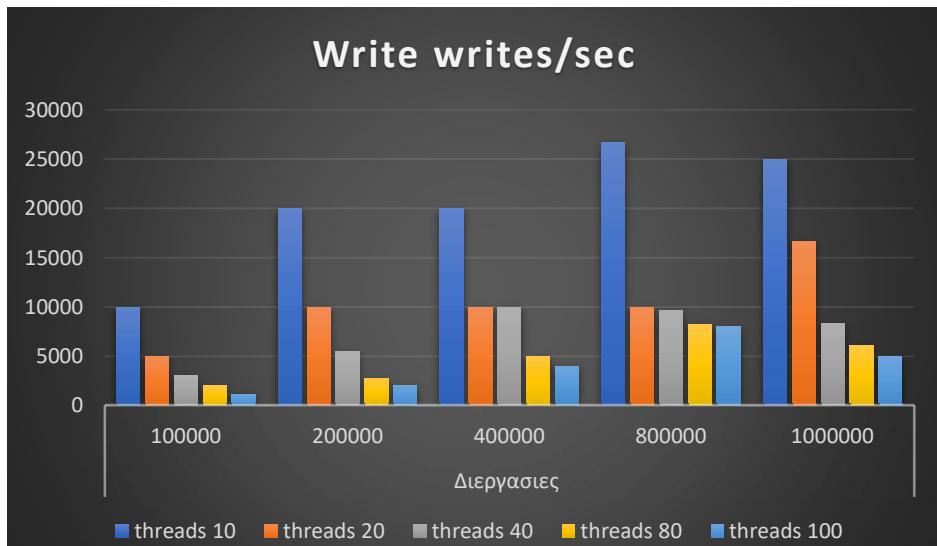
Write time(sec)		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	10	0.0001	0.00005	0.00005	0.000037	0.00004
	20	0.0002	0.0001	0.0001	0.0001	0.00006
	40	0.000333	0.000183	0.0001	0.000104	0.00012
	80	0.0005	0.000367	0.0002	0.000121	0.000163
	100	0.000867	0.000483	0.00025	0.000125	0.0002



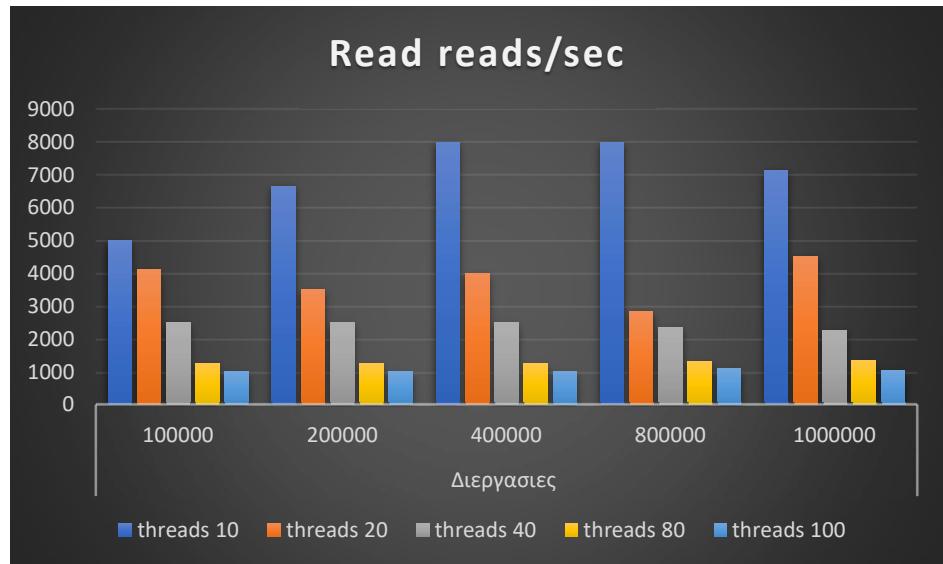
Read time(sec)		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	10	0.0002	0.00015	0.000125	0.000125	0.00014
	20	0.000243	0.000286	0.00025	0.00035	0.000221
	40	0.0004	0.0004	0.0004	0.000425	0.00044
	80	0.0008	0.0008	0.0008	0.000759	0.000744
	100	0.001	0.001	0.000986	0.000909	0.000954



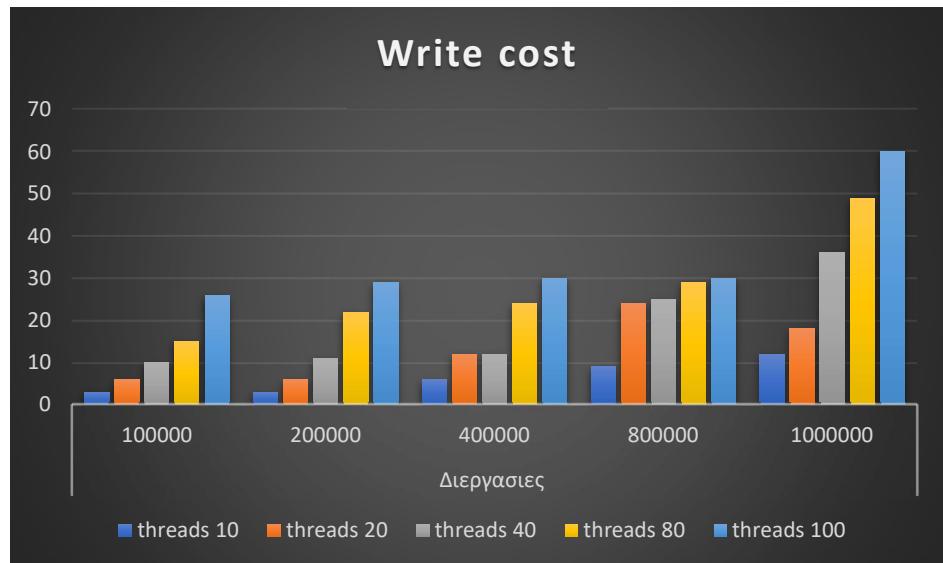
Write writes/sec		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	10	10000	20000	20000	26666.7	25000
	20	5000	10000	10000	10000	16666.7
	40	3000	5454.5	10000	9600	8333.3
	80	2000	2727.3	5000	8275.9	6122.4
	100	1153.8	2069	4000	8000	5000



Read reads/sec		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	10	5000	6666.7	8000	8000	7142.9
	20	4117.6	3500	4000	2857.1	4516.1
	40	2500	2500	2500	2352.9	2272.7
	80	1250	1250	1250	1317.6	1343.6
	100	1000	1000	1014.5	1100.2	1047.9



Write cost		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads threads	10	3	3	6	9	12
	20	6	6	12	24	18
	40	10	11	12	25	36
	80	15	22	24	29	49
	100	26	29	30	30	60



Read cost		Διεργασίες				
		100000	200000	400000	800000	1000000
Threads	10	14	21	35	70	98
	20	17	40	70	196	155
	40	35	56	112	238	308
	80	56	112	224	425	521
	100	70	140	276	509	668

