ΛΕΤΣΟΣ ΟΔΥΣΣΕΑΣ 2745
ΣΤΑΒΑΡΑΚΗΣ ΠΑΝΑΓΙΩΤΗΣ 2821
ΚΑΡΑΝΙΚΑ ΕΦΗ 2453

Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης δεδομένων

Στην εργασία αυτή μας ζητήθηκε να μετατρέψουμε μια έτοιμη μηχανή αποθήκευσης δεδομένων χρησιμοποιώντας πολυνηματισμό
για την εξασφάλιση γρηγορότερων αποτελεσμάτων.

Ουσιαστικά θα υλοποιήσουμε μια πολυνηματική λειτουργεία των εντολών put και get που διαθέτει η μηχανή.

Συνοπτικά, για την υλοποίηση των παραπάνω , αρχικά θα κατανοήσουμε πως λειτούργει η μηχανή αποθήκευσης που μας δόθηκε και στην συνέχεια θα χρησιμοποιήσουμε την pthreads βιβλιοθήκη ώστε να δημιουργήσουμε νήματα ξεχωριστά για τις λειτουργείες put και get έτσι ώστε να πετύχουμε τον ταυτοχρονισμό που μας ζητήθηκε.

Βασική προυπόθεση ώστε να μπορέσουμε να επιτύχουμε τον πολυνηματισμό ήταν να μπορέσουμε να δόσουμε πρόσβαση σε κάθε νήμα στην βάση δεδομένων της μηχανής αποθήκευσης , ώστε να μπορεί κάθε νήμα να επεξεργάζεται και να τροποποιεί τα δεδομένα της.

Προκειμένου να αποφύγουμε συγκρούσεις κατά την είσοδο πολλαπλών νημάτων στην βάση θα πρέπει να χρησιμοποιήσουμε (locks) , κλειδαρίες ώστε να ελέγχουμε καλύτερα την πρόσβαση των νημάτων στην βάση πετυχαίνωντας τον αμοιβαίο αποκλεισμό στις κρίσιμες περιοχές.

ΚΑΤΑΝΟΗΣΗ ΛΕΙΤΟΥΡΓΕΙΑΣ ΤΗΣ ΜΗΧΑΝΗΣ ΑΠΟΘΗΚΕΥΣΗΣ KIWI

Στην έτοιμη υλοποίηση που μας δόθηκε , προκειμένου να μπορέσουμε να κάνουμε μια εγγραφή write ή μια read ανάγνωση,
εφόσον είχαμε κάνει compile των κώδικα που μας δόθηκε , χρησιμοποιώντας την make all για την επικοινωνία όλων των αρχείων και αρχείων κεφαλίδων που μας δόθηκε εκτελέσαμε τις παρακάτω εντολές:

./**kiwi-bench write 100000**
./kiwi-bench read 100000

Ας ξεκινήσουμε με την λειτουργία write μιας και αντίστοιχη ροή έχει και η λειτουργεία read.

Η μηχανή μας αποτελείτε από 2 φακέλους : τον bench που είναι υπεύθυνος για την αλληλεπίδραση με το Terminal και την engine που είναι υπεύθυνη και για την εσωτερική λειτουργεία της μηχανής

Ας ξεκινήσουμε από τον φάκελο bench:

Bench.c:

Ξεκινώντας, την μηχανή συναντάμε την main συνάρτηση που βρίσκεται στο αρχείο bench.c.
Στην δεδομένη στιγμή δεν υπάρχει πολυνηματισμός , υπάρχει μόνο το αρχικό νήμα που διατρέχει τις συναρτήσεις της λειτουργείας write και read
Εκεί γίνεται και ο έλεγχος οτι πρόκειται για μια λειτουργεία write (η αντίστοιχα read) με βάση το όρισμα της γραμμής εντολών.
Έπειτα καλέιται η συνάρτηση write_test που βρίσκεται στο αρχείο kiwi.c

kiwi.c:

Μέσα στο αρχείο kiwi.c υπάρχει η write_test και read_test , αντίστοιχα.

Ενδεικτικά θα μιλήσουμε για την write_test, αντίστοιχη ροή ακολουθεί και η read_test.

Κατά την κλήση της η write_test αρχίκα ανοίγει την βάση δεδομένων (db= db_open(DATAS)) από το αρχείο db.c , και έπειτα χρησιμοποιεί την συνάρτηση get_ustime() για να μετρήσουμε τον χρόνο που θα χρειαστούμε για την λειτουργεία put.Στην συνέχεια επαναληπτικά βάζει τιμές στο ζεύγος key,value, και τελικά κάνει εγγραφή στην βάση (put) , με την εντολή db_add(db,&sk,&sv), συνάρτηση που και αυτή υλοποιείται στο db.c (του engine).Εκεί αξίζει να σημειωθεί η διαδικασία που ακολουθείτε στο memtable. Αφού ολοκληρωθούν οι εγγραφέςκλείνει η βάση, καλώντας την db_close(db) και κρατάει πάλι τον χρόνο ώστε με μια απλή αφαίρεση να δούμε τον χρόνο που χρειάστηκε για τις εγγραφές.

db.c:

Μέσα στο αρχείο db.c υλοποιούνται οι συναρτήσεις που πραγματοποιούν οποιαδίποτε ενέργεια πάνω στην βάση της μηχανής.Δηλαδή στην δικιά μας περίπτωση put με την συνάρτηση db_add ή get με την db_get , καθώς επίσης και τις συναρτήσεις db_open , db_close που ανοίγουν και κλείνουν την βάση αντίστοιχα, στην οποία επιτρέπουν να γίνονται εγγραφές και αναγνώσεις δεδομένων.Επιπλέον η βάση db.c είναι η διεπαφή μεταξύ των 2 δομών sst και memtable.

Memtable.c:

Σε αυτό το αρχείο υλοποιούνται οι απαραίτητες συναρτήσεις , προκειμένου να δούμε αν χρείαζεται να γίνει κάποιο compaction, καθώς επίσης και συναρτήσεις ώστε να γίνει reset το memtable.

sst.c:

Σε αυτό το αρχείο υλοποιοούνται οι συναρτήσεις που χρησιμοποιούνται για την εγγραφή-αφαίρεση αρχείων στον δίσκο,
καθώς επίσης και όταν το memtable γεμίσει υπάρχουν οι συναρτήσεις ώστε σε επικοινωνία με το memtable να γίνουν merge οι εγγραφές σύμφωνα με την συνάρτηση (sst_merge) στις και κατά την διαδικασία της αναζήτησης την (sst_get).

Αντιστοίχως υπάρχουν και τα αρχεία κεφαλίδων db.h , memtable.h , sst.h για τον engine φάκελο

Έπειτα θα αναλύσουμε την λειτουργεία της μηχανής στον engine φάκελο.

Κατά την λειτουργεία get:
Αρχικά γίνεται αναζήτηση για τον αν υπάρχουν εγγραφές στην κύρια μνήμη memtable και αν δεν βρεθεί κάτι , γίνεται αναζήτηση και στον SST που αντιστοιχεί σε πίνακα στον σκληρό δίσκο.

Κατά την λειτουργεία put:

Στην λειτουργεία put κατά την οποία προσθέτουμε εγγραφές στην μηχανή με την db_add() , αρχικά θα πρέπει να γίνει ένας έλεγχος , για το αν είναι αναγκαίο να γίνει compaction δηλαδή αν οι εγγραφές που θέλουμε να κάνουμε add χωράνε στον memtable ή αν προκαλούνε υπερχείληση.

Περίπτωση 1: Στην απλή περίπτωση που οι εγγραφές χωράνε στον memtable απλώς προχωράμε σε db_add προσθέτωντας τις εγγραφές μας.

Περίπτωση 2: Στην περίπτωση όμως που οι εγγραφές δεν χωράνε , χρειάζεται να γίνει compaction. Δηλαδή χρησιμοποιώντας ένα νήμα που εκτελείται παράλληλα οι εγγραφές του memtable να γίνουν merge στον sst και να γίνει reset ο memtable ώστε να χωράει τις ζητούμενες εγγραφες. Καθώς επίσης υπάρχει και ενα log αρχείο η χρησιμότητα του οποίου είναι να αποθηκεύει προσωρινά τα δεδομένα, έτσι ώστε σε περίπτωση εγγραφής,να αντιγράφονται εκτός απο το memtable και στο log.

ΒΕΛΤΙΩΣΕΙΣ ΚΑΙ ΑΛΛΑΓΕΣ ΓΙΑ ΠΟΛΥΝΗΜΑΤΙΣΜΟ

Προκειμένου να πετύχουμε τον πολυνηματισμό χρειάστηκε να τροποποιήσουμε τον κώδικα που μας δόθηκε ως εξής:

Αρχικά ένα σημαντίκο βήμα ήταν να δημιουργήσουμε τα απαραίτητα νήματα έτσι λοιπόν αρχικά κάναμε κάποιους ελέγους για να διαπιστώσουμε ποιές είναι οι λειτουργείες που ζητούνται και αν δίνονται τα σωστά ορίσματα στην γραμμή εντολών , έτσι ο αποδεκτός τρόπος να περάσουμε ορίσματα είναι μόνο ο

εξής<< ./kiwi-bench “λειτουργεία” αριθμός εγγραφών αριθμός νημάτων>>.Όπου αριθμός λειτουργεία έιναι μόνο ‘’write’’, ‘’read’’ ή ‘’readwrite’’ (μια επιπλέον λειτουργία που προσθέσαμε για το μείγμα λειτουργιών put και get), και τα άλλα 2 ορίσματα μόνο ακέραιοι αριθμοί.
Εν συνεχεία , ανάλογα την επιλογή λειτουργείας , σε κάθε λειτουργεία δημιουργήσαμε επαναληπτικά κάποια νήματα έτσι ώστε να μπορούν να εκτελούν τις λειτουργείες put και add παράλληλα.
Σε αυτό το σημείο συναντήσαμε την εξής δυσκολία , κατά την λειτουργεία write οδηγηθήκαμε σε segmatation fault,αυτό το πρόβλημα το επιλύσαμε ως εξής έγινε μια σημαντική αλλάγη στις συναρτήσεις write_test και read_test και αυτή ήταν ότι πλέον οι συναρτήσεις αυτές δεν μπορούσαν να κάνουν open_db και close_db γιατί τις χρησιμοποιούσε κάθε νήμα και είχε ως αποτέλεσμα κάποιο νήμα πχ να κλείνει την βάση ενώ κάποιο άλλο ακόμα πραγματοποιούσε αλλαγές.

Η λύση μας ήταν να εκτελέσουμε 1 φορά στην main την db_open ,έπειτα να εκτελέσουν παράλληλα ότι ενέργεια εκτελούν και τέλος πάλι να κλείσουμε την βάση στην main αφού όλα τα νήματα έχουν κάνει τις απαραίτητες επεξεργασίες στην βάση.

Αμέσως μετά έπρεπε με κάποιον τρόπο να διασφαλίσουμε ότι δεν θα υπάρχουν συγκρούσεις κατά την παράλληλη εκτέλεση των νημάτων.Αυτό το επιτύχαμε με την χρήση κλειδαριών (locks)στις κρίσιμες περιοχές που θεωρήσαμε έτσι ώστε να πετύχουμε αμοιβαίο αποκλεισμο.

Έπειτα χρειάστηκε να επιτύχουμε τον απαραίτητο συγχρονισμό μεταξύ των στατιστικών απόδοσης της κάθε λειτουργίας με διαφορετικά νήματα , όπου επίσης χρησιμοποιηθήκαν κλειδαριές για την επίτευξη πάλι του αμιβαίου αποκλεισμού διότι τα νήματα που τρέχουν σε κάθε λειτουργία προκαλούσαν συγκρούσεις στους χρόνους.**

Τέλος έχουμε δημιουργήσει 3 λειτουργίες όσον αφορά το μείγμα λειτουργιών:
1ή λειτουργια : Έχουμε 50% get , 50% put.
2η λειτουργία : Έχουμε 10% get, 90% put.
3η λειτουργια: Έχουμε 60% get, 40% put.

Σκοπός μας ήταν να δείξουμε όλους τους πιθανούς συνδιασμούς οσον αφορά τις προταιρεότητες.

ΠΡΟΒΛΗΜΑ ΑΝΑΓΝΩΣΤΩΝ-ΓΡΑΦΕΩΝ:

Εδώ αντιμετωπισαμε το εξής πρόβλημα,έπρεπε να διαχειριστούμε τον συγχρονισμό ανάμεσα σε γραφείς και αναγνώστες έτσι ωστε να μην υπάρξουν προβλήματα και συγκρούσεις.Πολλοί αναγνώστες μπορούν να έχουν ταυτόχρονη πρόσβαση στην βάση από τα νήματα που τους διατρέχουν , παρόλα αυτά, δεν είναι δυνατόν πολλοί γραφείς και ένας αναγνωστης ή πολλοί αναγνώστες και ένας γραφέας να έχουν πρόσβαση στην βάση συμφωνα με τα νήματα που έχουμε υλοποιήσει για την κάθε λειτουργία ετσι ώστε να πετύχουμε τον ταυτοχρονισμό που χρειαζόμαστε
Η επίλυση του προβλήματος ήρθε ως εξης:
ενας γραφεας θα πρεπει να εχει πρόσβαση στην βάση εκείνη την στιγμή ενώ κανένας αναγνώστης δεν θα μπορεί να διαβάσει απο αυτή καθώς επίσης και κανένας αναγνώστης δεν μπορεί να γραψει σε αυτη. Αντιθέτως πολλοί αναγνώστες μπορεί να εχουν ταυτοχρονη προσβαση στην βαση . Η υλοποιηση αυτη έγινε με την χρήση κλειδαριών (locks) αντίστοιχα σε κάθε μια λείτουργια db_add και db_get.

ΠΑΡΑΚΑΤΩ ΑΝΑΦΕΡΟΥΜΕ ΑΝΑΛΥΤΙΚΗ ΕΠΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ ΠΟΥ ΤΡΟΠΟΠΟΙΗΣΑΜΕ

bench.c:

1) Από γραμμή 1 κάνουμε #include τα

```
1 #include "bench.h"
2 #include <pthread.h>//βιβλιοθηκη για νηματα 1
3 #include "../engine/db.h"
4 #define DATAS ("testdb")
5
6 //orizw thn vash
7 DB* db;
```

απαραίτητα αρχεία κεφαλίδων και βιβλιοθήκες, pthread.h για τα νήματα, την . ./engine/db.h που χρειαζόμαστε για να ορίσουμε την μεταβλητή db που θα χρειαστεί να καλέσουμε αργότερα για να ανοίξουμε και να κλείσουμε την βαση μας. Και στην γραμμή 7 ορίζω DB* db;

```
84 //orizw sunarthsh pou tha pairnei san orisma to struct me to count to r kai ta threads gia thn read
85 void * read_thread(void* arg){
86         struct data *dataset=(struct data *) arg;
87         //pernaw sthn synarthsh kai ta threads pou thelw
88         _read_test(dataset->count,dataset->r,dataset->threads);
89         return 0;
90 }
91
92 //orizw sunarthsh pou tha pairnei san orisma to struct me to count to r kai ta threads gia thn write
93 void * write_thread(void* arg){
94         struct data *dataset=(struct data *) arg;
95         _write_test(dataset->count,dataset->r,dataset->threads);
96         return 0;
97 }
```

2)Στην γραμμή 84-97 ορίζουμε 2 συναρτήσεις με όρισμα το struct dataset με πεδία το count το r και τα threads έτσι ώστε να περάσουμε την συναρτηση read_test και

write_test του kiwi.c μέσα από τα νήματα που θα του δημιουργήσουμε αντιστοίχως.

```
104 //dhmiourgw sunarthsh etsi wste na ektypwnw tou xronous gia thn kathe mia leitoyrgia
105 void printer(char* action,long int count, struct  data threads_args ){
106
107     if (strcmp(action,"write") == 0) { //an h leitourgia einai write
108         printf("SELECTED ACTION: __WRITE__\n");
109         printf(LINE);
110         printf("|Random-WRITE    (done:%ld, found:%ld): %.6f sec/op; %.1f writes per sec(estimated); cost:%.3f(sec)\n",threads_args.count, count,
111         (double)(costofwrites / threads_args.count),(double)(threads_args.count / costofwrites),costofwrites);
112
113     }else if (strcmp(action,"read")==0){ //an h leitourgia einai read
114         printf("SELECTED ACTION:__READ__\n");
115         printf(LINE);
116         printf("|Random-READ    (done:%ld, found:%ld): %.6f sec/op; %.1f reads per sec(estimated); cost:%.3f(sec)\n",threads_args.count, count,(double)(costofreads / threads_args.count),
    (double)(threads_args.count / costofreads),costofreads);
117     }
118
119 }
120
```

3)Εδώ δημιουργόυμε μία συνάρτηση που καλείτε σε κάθε λειτουργία έτσι ώστε να εκτυπώνονται τα στατιστικά απόδοσης από την κάθε λειτουργια είτε ειναι read έιτε write. Ως ορισμα παίρνει την κάθε λειτουργία ξεχωριστά και ο struct που αναφέραμε προηγουμένος στου οποίου τα πεδία δίνουμε τιμες απο την κάθε περίπτωση που εχουμε υλοποιήσει (105-117)
Η συνάρτηση αυτή θα καλεστεί σε καθε περίπτωση που εχουμε υλοποιησει.

```
121 int main(int argc,char** argv)
122 {
123         //arxikopoiw tiw leitourgies
124         long int count,countofwrites,countofreads;
125
126         //arxikopoiw ta threads
127         long int threads,threadsofwrites,threadsofreads;
128
129         //dhlvsh gia thn epilogh
130         int epilogh=0;
131
132         //arxikopoiw ta struct pou xreiazomai
133         struct data threads_args,threads_args_writes,threads_args_reads;
134
135         srand(time(NULL));
136
137         //thelw na dinei o xrhsths nhmata ara arc<4
138         if (argc < 4) {
139                 fprintf(stderr,"Usage: db-bench <write | read | readwrite> <count> <threads> <random>\n");
140                 exit(1);
141         }
142
143         int r = 0;
144
145         //elegxos apo thn grammh entolwn
146         if ((strcmp(argv[1],"write") != 0) && (strcmp(argv[1],"read") != 0) && (strcmp(argv[1],"readwrite") != 0)){
147         fprintf(stderr,"Usage: db-bench <write | read |readwrite> <count> <threads> <random>\n");
148             exit(1);
149
150         }
```

4)Από την γραμμή 121 και έπειτα ξεκινάει η main συνάρτηση.
Στις γραμμές 124 αρχικοποιούμε τις λειτουργίες long int countofwrites,countofreads τις οποίες χρειαζόμαστε ετσι ώστε να φτιαξουμε τις μετρήσεις για κάθε ποσοστό που εχουμε υλοποιησει είτε για την read ειτε για την write.

Ενώ στην 127 αρχικοποιούμε τα νήματα για την write και τα read με την μεταβλητή threads ξεχωριστα , και threadsofwrites και threadsofreads για τα ποσοστα που εχουμε υλοποιήσει.
Στην γραμμή 133 αρχικοποιούμε τα απαραίτητα struct για την δημιουργία των νημάτων αντίστοιχα για την κάθε μια περίπτωση.
Έπειτα από την γραμμή 138 – 148 γίνεται ο έλεγχος για το αν έχουμε λάβει σωστά ορίσματα από την γραμμή εντολών.

```
152         //elegxos gia thn kathe mia periptwsh read,write h readwrite
153         switch(strlen(argv[1])){
154
155                 case 5://periptwsh pou einai write h leitourgia
156
157                         count = atoi(argv[2]);
158                         _print_header(count);
159                         _print_environment();
160
161
162                         if (argc == 5)
163                                 r = 1;
164
165                         //pairnw to threads apo thn grammh entolwn
166                         threads=atoi(argv[3]);
167
168                         //arxikopoiw ta nhmata gia to write me malloc
169                         pthread_t *tidofwrite = (pthread_t*)malloc(threads * sizeof(pthread_t));
170
171                         //anoigw vash
172                         db = db_open(DATAS);
173
174                         //dinw sto struct ta orismata
175                         threads_args.count=count;
176                         threads_args.r=r;
177                         threads_args.threads=threads;
178
179                         //dhmiourgw ta nhmata
180                         for(int i=0;i<threads;i++){
181                                 pthread_create(&tidofwrite[i],NULL,write_thread,&threads_args);
182                         }
183                         for(int i=0;i<threads;i++){
184                                 pthread_join(tidofwrite[i],NULL);
185                         }
186
187                         //apodesmeuw gia thn malloc
188                         free(tidofwrite);
189
190                         //kleinw vash
191                         db_close(db);
192
193
194                         //ektypwnw ta statistika apodoshs ths leitourgias write
195                         printer("write",count,threads_args);
196
```

Στην γραμμή 152 ορίζουμε μια δομή switch – case για τα 3 διαφορετικά cases λειτουργιών.
Έτσι στην γραμμή 155 έχουμε το case για την λειτουργία write
157-169 παίρνω τα ορίσματα από την γραμμή εντολών και δεσμεύω τον απαραίτητο χώρο για τα threads μέσω της λειτουργίας malloc.
Γραμμή 172 ανοίγουμε την βάση db=db_open(DATAS) και την αποθηκεύουμε στην μεταβλητή db που ορίσαμε παραπανω.
Από 174- 186 δημιουργώ τα νήματα μέσω της int pthread_create (pthread_t *thread, const pthread_attr_t*attr, void *(routine)(void *), void*arg); και ετσι ώστε τα νήματα να τρέξουνε την write_thread που ορίσαμε παραπάνω για κάθε ενα νήμα που δίνει ο χρηστηςαπο την γραμμή εντολών, έπειτα τερματίζω τα νήματα μεσω της int pthread_join (pthread_t thread, void **status); αποδεσμεύοντας τους πόρους του κάθε νήματος.
στην 188 αποδεσμεύω την μνήμη που ειχαμε δεσμεέυσει free(tidofwrite).Τέλος στην 191 αφού τα νήματα εχουν ολοκληρωσει τις διεργασίες τους , κλείνω την βαση μεσω της db_close(db), και καλώ την printer ώστε να εκτυπώσει τα απαραίτητα αποτελέσματα.
\

```
204
205
206             case 4:
207
208                     count = atoi(argv[2]);
209
210
211                     _print_header(count);
212                     _print_environment();
213
214
215                     if (argc == 5)
216                             r = 1;
217
218                     //pairnw to threads apo thn grammh entolwn
219                     threads=atoi(argv[3]);
220                     //desmeyw xwro gia ta thread ths read
221                     pthread_t *tidofread = (pthread_t*)malloc(threads * sizeof(pthread_t));
222
223                     //anoigw vash
224                     db = db_open(DATAS);
225
226                     //dinw sto struct ta orismata
227                     threads_args.count=count;
228                     threads_args.r=r;
229                     threads_args.threads=threads;
230
231                     //dhmioyrgw ta nhmata gia thn read
232                     for(int i=0;i<threads;i++){
233                             pthread_create(&tidofread[i],NULL,read_thread,&threads_args);
234                     }
235                     for(int i=0;i<threads;i++){
236                             pthread_join(tidofread[i],NULL);
237                     }
238
239                     //apodesmeyw ton xwro apo thn malloc
240                     free(tidofread);
241
242                     //kleinw vash
243                     db_close(db);
244
245                     //ektypwnw ta statika xronou ths leitoyrgias read
246                     printer("read",count,threads_args);
247
248
249                     //_read_test(count, r);
250                     break;
```

Στην ίδια λόγική με τα παραπάνω από την γραμμή 206 έως 250 , έχουμε τον κώδικα για την λειτουργία read.

```
254                 case 9:// periptwsh readwrite
255
256                         count = atoi(argv[2]);
257                         _print_header(count);
258                         _print_environment();
259
260
261                         if (argc == 5)
262                                 r = 1;
263
264                         //pairnw to threads apo thn grammh entolwn
265                         threads = atoi(argv[3]);
266
267                         //dinw ston xrhsth thn epilogh na epileksei stiw treiw epilogew pou exoume ulopoihsh gia to mei
268                         printf("If you want the 50 percect of put and 50 percent of get choose 1.If you want the 90 per
    put and the 60 percent of get choose 3.  \n");
269                         scanf("%d",&epilogh);
270
271
272                         //an epileksei epileksei 50-50
273                         if(epilogh==1){
274
275                                 //metrhths gia ta write 50
276                                 countofwrites=count*50/100;
277                                 //methrths gia ta read 50
278                                 countofreads=count*50/100;
279
280                                 //threads gia ta write
281                                 threadsofwrites=(long int)threads*50/100;
282                                 //threads gia ta read
283                                 threadsofreads=(long int)threads*50/100;
284
285                                 //deysmeyw xwro gia ta threads tou write kai tou read
286                                 pthread_t *tidofwrites = (pthread_t*)malloc(threads * sizeof(pthread_t));
287                                 pthread_t *tidofreads = (pthread_t*)malloc(threads * sizeof(pthread_t));
288
289                                 //anoigw vash
290                                 db = db_open(DATAS);
291                                 //twra einai me 50 write 50 read meta tha to allaksw
```

,Από την γραμμή 254 και μετά , έχουμε την 3η περίπτωση της λειτουργίας readwrite.

Έχουμε δημιουργήσει 3 επιλογές ανάλογα με το ποσοστό % απο τον κάθε τυπο put και get από τις οποίες ο χρήστης θα επιλέγει ποιά θα χρησιμοποιήσει μεσω της **printf**("**If you want the 50 percect of put and 50 percent of get choose 1.If you want the 90 percent put and 10 percent of get choose 2.If you want the 40 percent put and the 60 percent of get choose 3. \n");** και της **scanf("%d",&epilogh);**.

Έτσι στην πρώτη επιλογή (273 γραμμή) έχουμε επιλέξει 50% reads 50%writes. Επειτα στην 276-278 μοιράζουμε τις εγγραφές καθώς και τα νήματα σύμφωνα με το ποσοστό που έχουμε επιλέξει.’Επειτα δεμεύουμε δυναμικά τον χώρο που χρείαζονται τα νήματα για να δημιουργηθούν.(286-287) οπως κάναμε στην write και στην read αναλόγως για κάθε λειτουργία.

Έπειτα ανοίγουμε την βάση ώστε να μπορούν ταυτόχρονα τα νηματα να εχουν πρόσβαση και να την επεξεργαστουν.

```
292
293                     //struct gia ta write
294                     threads_args_writes.count=countofwrites;
295                     threads_args_writes.r=r;
296                     threads_args_writes.threads=threadsofwrites;
297
298                     //struct gia ta read
299                     threads_args_reads.count=countofreads;
300                     threads_args_reads.r=r;
301                     threads_args_reads.threads=threadsofreads;
302
303                     //dhmiourgw nhmata gia ta write kai gia read
304                     for(int i=0;i<threadsofwrites;i++){
305                             pthread_create(&tidofwrites[i],NULL,write_thread,&threads_args_writes);
306                     }
307                     for(int i=0;i<threadsofreads;i++){
308                             pthread_create(&tidofreads[i],NULL,read_thread,&threads_args_reads);
309                     }
310                     for(int i=0;i<threadsofwrites;i++){
311                             pthread_join(tidofwrites[i],NULL);
312                     }
313                     for(int i=0;i<threadsofreads;i++){
314                             pthread_join(tidofreads[i],NULL);
315                     }
316
317                     //apodeysmeyw malloc
318                     free(tidofwrites);
319                     free(tidofreads);
320
321                     //kleinw vash
322                     db_close(db);
323
324                     //ektypwnw statistika apodoshs antistoixa
325                     printer("read",countofreads,threads_args_reads);
326                     printer("write",countofwrites,threads_args_writes);
327
328
329
330                     break;
```

.

Στην 292 – 301 δημιουργούμε τα struct αντίστοιχα για κάθε μια λειτουργία οπως καναμε και παραπάνω για την read και την write 304- 315 δημιουργούμε τα νήματα αντοιστοιχα για read και write ανάλογα με το ποσοστό που έχει δώσει ο χρήστης και αποδεσμευουμε τους πόρους του νημάτων.Τελικά στις γραμμές 317-318 αποδευσμέουμε τον χωρο της malloc για κάθε ενα id νηματων και καλούμε 2 φορές την printer (325-326) , μια για τα read και μια για τα writes για την εκτυπωση των αποτελεσμάτων από τα στατιστικά απόδοσης.

```
366                             //dhmiourgw nhmata gia ta write kai gia read
367                             for(int i=0;i<threadsofwrites;i++){
368                                     pthread_create(&tidofwrites[i],NULL,write_thread,&threads_args_writes);
369                             }
370                             for(int i=0;i<threadsofreads;i++){
371                                     pthread_create(&tidofreads[i],NULL,read_thread,&threads_args_reads);
372                             }
373                             for(int i=0;i<threadsofwrites;i++){
374                                     pthread_join(tidofwrites[i],NULL);
375                             }
376                             for(int i=0;i<threadsofreads;i++){
377                                     pthread_join(tidofreads[i],NULL);
378                             }
379
380                             //apodeysmeyw malloc
381                             free(tidofwrites);
382                             free(tidofreads);
383
384
385                             db_close(db);
386
387
388
389                             //ektypwnw statistika apodoshs antistoixa
390                             printer("read",countofreads,threads_args_reads);
391                             printer("write",countofwrites,threads_args_writes);
392
393                             break;
360
361                             //struct gia ta read
362                             threads_args_reads.count=countofreads;
363                             threads_args_reads.r=r;
364                             threads_args_reads.threads=threadsofreads;
```

.Ομοίως έχουμε την επιλογή 2 για τα ποσοστα 90% writes και 10% reads.

```
395                     }else if(epilogh==3){
396
397                             //metrhths gia ta write 40
398                             countofwrites=count*40/100;
399                             //metrhths gia ta read 60
400                             countofreads=count*60/100;
401
402                             //threads gia ta write
403                             threadsofwrites=(long int)threads*40/100;
404                             //threads gia ta read
405                             threadsofreads=(long int)threads*60/100;
406
407                             printf("Threads of reads is %ld",threadsofreads);
408                             printf("Threads of writes is %ld \n",threadsofwrites);
409
410                             //deysmeyw xwro gia ta threads tou write kai tou read
411                             pthread_t *tidofwrites = (pthread_t*)malloc(threadsofwrites * sizeof(pthread_t));
412                             pthread_t *tidofreads = (pthread_t*)malloc(threadsofreads * sizeof(pthread_t));
413
414                             //anoigw vash
415                             db = db_open(DATAS);
416
417
418                             //struct gia ta write
419                             threads_args_writes.count=countofwrites;
420                             threads_args_writes.r=r;
421                             threads_args_writes.threads=threadsofwrites;
422
423                             //struct gia ta read
424                             threads_args_reads.count=countofreads;
425                             threads_args_reads.r=r;
426                             threads_args_reads.threads=threadsofreads;
```

.

```
428                     //dhmiourgw nhmata gia ta write kai gia read
429                     for(int i=0;i<threadsofwrites;i++){
430                             pthread_create(&tidofwrites[i],NULL,write_thread,&threads_args_writes);
431                     }
432                     for(int i=0;i<threadsofreads;i++){
433                             pthread_create(&tidofreads[i],NULL,read_thread,&threads_args_reads);
434                     }
435                     for(int i=0;i<threadsofwrites;i++){
436                             pthread_join(tidofwrites[i],NULL);
437                     }
438                     for(int i=0;i<threadsofreads;i++){
439                             pthread_join(tidofreads[i],NULL);
440                     }
441
442                     //apodesmeyw malloc
443                     free(tidofwrites);
444                     free(tidofreads);
445
446
447                     db_close(db);
448
449                     //ektypwnw statistika apodoshs antistoixa
450                     printer("read",countofreads,threads_args_reads);
451                     printer("write",countofwrites,threads_args_writes);
452
453                     break;
```

..Ομοίως έχουμε την επιλογή 3 για τα ποσοστα 40% writes και 60% reads.

!!!!!!Οι επιλογές των δοκιμασμένων ποσοστών ήταν για να καλύψουμε τις 3 περιπτωσεις : reads == write

reads< writes

reads>writes.!!!

bench.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <pthread.h>
#define KSIZE (16)
#define VSIZE (1000)

#define LINE "+-----------------------------+---------------+-----------------------------+------------------+\n"
#define LINE1 "-----------------------------------------------------------------------------------------------------\n"

long long get_ustime_sec(void);
void _random_key(char *key,int length);

//arxikopoiw struct etsi vste na parw ta orismata gia ta nhmata gia thn kathe leitourgia
struct data {
        long int count;
        int r;
        long int threads;
};

//arxikopoiw sunarthsh etsi wste na ektupwsw ta statistika xronou thw kathe leitourgias read kai write
void printer(char* action,long int count, struct  data threads_args);

//sunarthsh etsi vste ta nhmata na treksoun aythn me ta orismata pou tou exw orisei
void * write_thread(void* arg);
void * read_thread(void* arg);

//arxikopoiw sunarthsh pou ylopoioun to sunoliko kostow gia thn kathe leitourgia
double costwrites(double cost,double newcost);
double costreads(double cost,double newcost);

//arxikopoiw kleidaria gia reads kai writes
pthread_mutex_t mutexofreads;
pthread_mutex_t mutexofwrites;

//arxikopoiw antistoixa to kostos gia ta read kai ta write
double costofreads;
double costofwrites;
```

Το αρχείο bench.h ουσιαστικά είναι οι υπογραφές των συναρτήσεων που έχουμε χρησιμοποιήσει στο bench.c. και kiwi.c.

kiwi.c:

```
7  //orizw vash
8  DB *db;
9
10 //arxikopoiw mutex gia read kai write
11 pthread_mutex_t mutexofreads=PTHREAD_MUTEX_INITIALIZER;
12 pthread_mutex_t mutexofwrites=PTHREAD_MUTEX_INITIALIZER;
13
14 //dhmiourgw sunarthsh pou epistrefei to kostos apo ta writes
15 double costwrites(double cost,double newcost){
16         costofwrites=cost+newcost;
17         return costofwrites;
18 }
19 //dhmiourgw sunarthsh pou epistrefei to kostos apo ta reads
20 double costreads(double cost,double newcost){
21         costofreads=cost+newcost;
22         return costofreads;
23 }
```

.Γραμμές 7-12 ορίζουμε την βάση DB *db , και κάνουμε init τα mutexes ωστε να μπορέσουν τα πολλαπλα νηματα για τα συνολικα κόστη των read και write αντιστοιχως το οποιο θα υλοποιήσουμε αργοτερα ετσι ωστε να επιτύχουμε τον αμιβαίο αποκλείσμο διοτι πρόκειται για κρίσιμες περιοχές.

Έπειτα στίς γραμμές 14-23 δημιουργούμε 2 συναρτήσεις οι οποίες υπολογίζουν το συνολικό κόστος παίρνωντας ως όρισμα το νέο κοστος απο κάθε νημα και το παλιο κόστος από το προηγούμενο και το επιστρέφουν ωστε στην συνέχεια να αποθηκευτεί στην μεταβλητη costofwrites που εχει οριστεί στην bench.h.

```
25 //sunarthsh write test san epipleon orisma ta threads etsi wste na ta spasw analoga me tiw leitourgies
26 void _write_test(long int count, int r,long int threads)
27 {
28         int i;
29         double cost;
30         long long start,end;
31         Variant sk, sv;
32
33
34         char key[KSIZE + 1];
35         char val[VSIZE + 1];
36         char sbuf[1024];
37
38         memset(key, 0, KSIZE + 1);
39         memset(val, 0, VSIZE + 1);
40         memset(sbuf, 0, 1024);
41
42         start = get_ustime_sec();
43         //metrhths gia kathe leitorugia analoga me threads
44         long int counter=(long int)count/threads;
45
46         for (i = 0; i < counter; i++) {
47                 if (r)
48                         _random_key(key, KSIZE);
49                 else
50                         snprintf(key, KSIZE, "key-%d", i);
51                 fprintf(stderr, "%d adding %s\n", i, key);
52                 snprintf(val, VSIZE, "val-%d", i);
53
54                 sk.length = KSIZE;
55                 sk.mem = key;
56                 sv.length = VSIZE;
57                 sv.mem = val;
58
59                 db_add(db, &sk, &sv);
60                 if ((i % 10000) == 0) {
61                         fprintf(stderr,"random write finished %d ops%30s\r",
62                                         i,
63                                         "");
64
65                         fflush(stderr);
66                 }
67         }
68
69
70         end = get_ustime_sec();
71         cost = end -start;
```

Έπειτα στην γραμμή 26 έχουμε την write_test τροποποιημένη , προσθέτωντας σαν όρισμα τα νήματα στα οποία θα μοιραστούν οι εγγραφές.Στην γραμμή 44 έχουμε την μεταβλήτη counter που υπολογίζει πόσες εγγραφές αντισοιχούν σε καθε thread.

!!!!Στην γραμμή (76) έχουμε θεωρήσει κρίσημη περιοχή και έχουμε κάνει lock για την επιτευξη του αμοιβαίου αποκλεισμού ωστε να επιτυχουμε την αποφυγη συγκρούσεων των νημάτων στον υπολογισμό του κόστους σύμφωνα με την συνάρτηση: **pthread_mutex_lock(&mymutex);**

Στην γραμμή 78 ενώ εχουμε βάλει τις απαραίτητες κλειδαρίες καλούμε την costofwrites, η οποία μας επιστρέφει το συνολικό κόστος προστήθωντας το κόστος απο κάθε νημα στο συνολικό αποφεύγοντας τις συγκρούσεις.Άρα κάθε νήμα κάθε φορά θα μπαίνει και θα ενημερώνει το συνολικό κόστος μην δίνοντας προσβαση στα αλλα να μπουν την δεδομένη στιγμή.

Έπειτα κάνουμε **pthread_mutex_unlock(&mymutex);** ετσι ώστε να ξεκλειδώσουμε το mutex διότι εκτελέστηκε η κρίσιμη περιοχή.

```
74          //diaforetikes kleidaries gia kathe read kai write
75          //lock unlock krisimh perioxh
76          pthread_mutex_lock(&mutexofwrites);
77          //sunoliko kostos prosthetete se kathe nhma pou ylopoiei to write to neo kostos amoibaios apokleismous etsi e
    ananewsei ton xrono symfvna me ayto
78          costofwrites=costwrites(cost,costofwrites);
79          pthread_mutex_unlock(&mutexofwrites);
80
81
82
83  }
84  //sunarthsh read test epishs san epipleon orisma ta threads etsi wste na ta spasw analoga me tiw leitourgies
85  void _read_test(long int count, int r,long int threads)
86  {
87          int i;
88          int ret;
89          int found = 0;
90          double cost;
91          long long start,end;
92          Variant sk;
93          Variant sv;
94          char key[KSIZE + 1];
95          long int counter;
96
97
98
99          start = get_ustime_sec();
100         //metrhths gia kathe leitorugia analoga me threads
101         counter=(long int)count/threads;
102         for (i = 0; i < counter;i++) {
103                 memset(key, 0, KSIZE + 1);
104
105                 /* if you want to test random write, use the following */
106                 if (r){
107                         _random_key(key, KSIZE);
108                 }else{
109                         snprintf(key, KSIZE, "key-%d", i);
110                 }
111                 fprintf(stderr, "%d searching %s\n", i, key);
112                 sk.length = KSIZE;
113                 sk.mem = key;
114                 ret =db_get(db, &sk, &sv);
115                 if (ret) {
116                         //posa vrethikan
117                         //db_free_data(sv.mem);
118                         found++;
119                 } else {
120                         INFO("not found key#%s",
121                                         sk.mem);
122         }
```

Ομοίως με την write έτσι και εδώ από την γραμμή 85 έως 123, έχουμε την λειτουργία read.

```
136         cost = end - start;
137
138         //lock unlock krisimh peioxh amoibaios apokleismos gia kathe nhma gia to sunoliko kostow twn read
139         pthread_mutex_lock(&mutexofreads);
140         //apothikeyetai sthn metavlith costofreads ananewnetai kathe fora apo kath nhma ksexwrista
141         costofreads=costreads(cost,costofreads);
142         pthread_mutex_unlock(&mutexofreads);
143
144
145 }
```

Και εδω στην γραμμή 136 -145 ορίζουμε πάλι κρίσιμη περιοχή για τον υπολογισμό του κόστους των read πετυχαίνωντας αμοιβαιο αποκλεισμό όπως παραπάνω.

db.c

```
54 int addwritenum(int writer_num){
55         writer_num=writer_num +1;
56         return writer_num;
57 }
58 int subwritenum(int writer_num){
59         writer_num=writer_num-1;
60         return writer_num;
61 }
62
63
64
65
66 int db_add(DB* self, Variant* key, Variant* value)//epistrefei 1
67 {
68     pthread_mutex_lock(&self->writer);//eisodos se krisimh perioxh amoibaios apokleismos me locks
69
70      //vazw kleidi kai value sto memtable h sto sst ama xreiazetai compaction
71     int memtableadd=0;
72     if (memtable_needs_compaction(self->memtable))
73     {
74         INFO("Starting compaction of the memtable after %d insertions and %d deletions",
75              self->memtable->add_count, self->memtable->del_count);
76         sst_merge(self->sst, self->memtable);
77         memtable_reset(self->memtable);
78     }
79     memtableadd=memtable_add(self->memtable, key, value);
80
81
82     pthread_mutex_unlock(&self->writer);//eksodos apo krisimh perioxh
83     //apothikeyw thn epistrofh ths sunarthshs sthn metvlhth memtableadd epeidh to nhma ua prepei na dei thn synarthsh
metablhth wste to epomeno nhmana kalesei neasynarthsh
84     return memtableadd;
85 }
```

Στην γραμμή 66 έχουμε την db_add
Η αλλαγή που πραγματοποιήσαμε έτσι ωστε να πετυχουμε τον ταυτοχρονισμο ενας γραφέας πολλόι αναγνώστες που αναφέραμε παραπάνω ηταν να προσθέσουμε κλειδαρίες (locks) για το add και για το get.Συγκεκριμένα έχουμε αρχικοποιήσει στην db.h τα 2 mutexes που χρειαστήκαμε.Στην γραμμή 68 με σκοπό να αποφύγουμε την σύγκρουση έχουμε χρησιμοποιήσει **pthread_mutex_lock(&self→writer);(**68) για την είσοδο σε κρίσιμη περιοχή διοτι όταν θελουμε να εκτελέιται οι εγγραφες θα πρέπει ένα νήμα καθε φορα απο αυτα που χρησιμοποιούν τα write να τις υλοποιει αφήνοντας απέξω τα αλλα.
Μόλις ολοκληρωθούν οι εγγραφές απο κάθε νήμα τότε συμφωνα με την συνάρτηση **pthread_mutex_unlock(&self→writer);** (82)βγαίνουμε απο την κρίσιμη περιοχή έχοντας εκτελεστεί ο κώδικας αυτής.
Επίσης στην γραμμή 71 έχω ορίσει την μεταβλητή memtableadd την οποία την χρησιμοποιώ αργότερα ετσι ωστε να αποθηκεύσω την τιμη της συναρτησης memtable_add(self->memtable, key, value) σε αυτήν , επειδή το επόμενο νήμα μπορεί να έχει ήδη ξεκινήσει και δεν πρέπει να χαθεί αυτη η τιμή.

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

    pthread_mutex_lock(&self->reader);//amoivaios apokleismos
    self->readers_num=addreadnum(self->readers_num);//prosthetw wste na exw to poly enan reader wste na mhn afhsw kapoion writer na
    if (self->readers_num==1)//lock ama vrei kapoion writer mesa sto read
                pthread_mutex_lock(&self->writer);
    pthread_mutex_unlock(&self->reader);

    int sstget=0;
    //paei prwta elegxei sto memtable an den brethei
    if (memtable_get(self->memtable->list, key, value) == 1)
        sstget=1;
    //kanei anazhthsh sto sst kai to epistrefei
    sstget=sst_get(self->sst, key, value);


    pthread_mutex_lock(&self->reader);//eisodos se krisimh perioxh
    self->readers_num =subreadnum(self->readers_num);//afairw enan reader mexris otoy ginei 0 kai afhnw touw writes na treksoun
    if (self->readers_num==0)//vgainei apo readers kai afhnei writers na treksoun
                pthread_mutex_unlock(&self->writer);
    pthread_mutex_unlock(&self->reader);//eksodos apo krisimh perioxh

    return sstget;
}
```

Προκειμένου να δώσουμε πρόσβαση σε πολλούς αναγνώστες αρχικά προσθεσαμε στην γραμμη 102 ένα lock **pthread_mutex_lock(&self->reader);** για να βάλουμε σε αναμονή το συγκεκριμένο νήμα ενώ τα αλλα περιμένουν απέξω.
Βρισκομαστε σε reader αρα προσθετουμε 1 reader συμφωνα με την addreadnum(int readers_num)(89) οποτε υπάρχει τουλάχιστον ένας αναγνώστης διασφαλιζοντας μας οτι κανενας writer δεν μπορεί να εισέλθει.Σε περιπτωση που εντοπισουμε writer μέσα στον reader κανουμε **pthread_mutex_lock(&self→writer);** για να κρατήσουμε σε αναμονή τον writer .και επειτα βγαινουμε απο την κρισιμη περιοχη καθε φορα με το pthread_mutex_unlock(&self->reader) της γραμμης (106) Επειτα ξαναμπαίνουμε σε κρίσιμη περιοχή και αφαιρω όσο υπαρχουν 1 προς 1 τους readers με την **int subreadnum(int readers_num)**(117) μεχρι να γίνουν 0.Πλέον έχοντας τελειώσει με τους readers κάνω unlock τους writers(119) ωστε να μπορέσουν να εκτελεστούν τα puts και τελος κανω unlock τους reader (120) γιατι βγαινουμε απο την κρισιμη περιοχη.

Παρακάτω παραθέτουμε τα στατιστικά από τα πειράματα μας:

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make clean
cd engine && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
rm -rf *.o libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
rm -f kiwi-bench
rm -rf testdb
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
    CC db.o
    CC memtable.o
    CC indexer.o
    CC sst.o
    CC sst_builder.o
    CC sst_loader.o
    CC sst_block_builder.o
    CC hash.o
    CC bloom_builder.o
    CC merger.o
    CC compaction.o
    CC skiplist.o
    CC buffer.o
    CC arena.o
    CC utils.o
    CC crc32.o
    CC file.o
    CC heap.o
    CC vector.o
    CC log.o
    CC lru.o
    AR libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
gcc -g -ggdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -lpthread -lsnappy -o kiwi-bench
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$ cd bench/
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 100000 5
```

```
SELECTED ACTION: __WRITE__
+-----------------------------+---------------+-----------------------------+-----------------+
|Random-WRITE     (done:100000, found:100000): 0.000050 sec/op; 20000.0 writes per sec(estimated); cost:5.000(sec)
```

.

WRITE

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	0.000050	0.000100	0.000200	0.000490	0.000980
200000	-	0.000050	0.000095	0.000195	0.000515	0.000985
400000	-	0.000060	0.000135	0.000262	0.000607	0.001080
800000	-	0.000081	0.000125	0.000275	0.000674	0.001238
1000000	-					

.

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	20000.0	10000.0	5000.0	2040.8	1020.4
200000	-	20000.0	10526.3	5128.2	1941.7	1015.2
400000	-	16666.7	7407.4	3809.5	1646.1	925.9
800000	-	12307.7	8000.0	3636.4	1484.2	808.1
100000	-					

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	5.000	10.000	20.000	49.000	98.000
200000	-	10.000	19.000	39.000	103.000	197.000
400000	-	24.000	54.000	105.000	243.000	432.000
800000	-	65.000	100.000	220.000	539.000	990.000
100000	-					

.

READ

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	0.000000	0.000000	0.000000	0.000000	0.000190
200000	-	0.000050	0.000050	0.000000	0.000000	0.000000
400000	-	0.000037	0.000050	0.000087	0.000158	0.000290
800000	-	0.000025	0.000037	0.000075	0.000126	0.000321
1000000	-					

.

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	inf	inf	inf	inf	5263.2
200000	-	20000.0	20000.0	inf	inf	inf
400000	-	26666.7	20000.0	11428.6	6349.2	3448.3
800000	-	40000.0	26666.7	13333.3	7920.8	3112.8
100000	-					

.

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
100000	-	0.000	0.000	0.000	0.000	19.000
200000	-	10.000	10.000	0.000	0.000	0.000
400000	-	15.000	20.000	35.000	63.000	116.000
800000	-	20.000	30.000	60.000	101.000	257.000
100000	-					

READWRITE

50% PUT-50% GET(ΔΟΚΙΜΗ ΓΙΑ 100000 ΛΕΙΤΟΥΡΓΙΕΣ ΚΑΙ ΓΙΑ 200000 ΛΕΙΤΟΥΡΓΙΕΣ READWRITE)

WRITE

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	0.000040	0.000200	0.000200	0.000480	0.001000
100000	-	0.000180	0.000120	0.000200	0.000440	0.001030

READ

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	0.000040	0.000120	0.000200	0.000480	0.000920
100000	-	0.000160	0.000120	0.0.000200	0.000320	0.001280

WRITE

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	25000.0	5000.0	5000.0	2083.3	1000.0
100000	-	6250.0	8333.3	5000.0	3125.0	970.9

READ

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	25000.0	8333.3	5000.0	2083.3	1087.0
100000	-	5555.6.0	8333.3	5000.0	2272.7	781.2

WRITE

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	2.000	10.000	10.000	24.000	50.000
100000	-	16.000	12.000	20.000	44.000	103.000

READ

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
50000	-	2.000	6.000	10.000	24.000	46.000
100000	-	18.000	12.000	20.000	32.000	128.000

READWRITE

40% PUT-60% GET(ΔΟΚΙΜΗ ΓΙΑ 100000 ΛΕΙΤΟΥΡΓΙΕΣ ΚΑΙ ΓΙΑ 200000 ΛΕΙΤΟΥΡΓΙΕΣ READWRITE)

WRITE

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
40000	-	0.000100	0.000183	0.000200	0.000400	0.000925
80000	-	0.000075	0.000100	0.000175	0.000500	0.001000

READ

ΡΥΘΜΑΠΟΔΟΣΗ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
60000	-	0.000083	0.000200	0.000200	0.000400	0.000883
120000	-	0.000075	0.000100	0.000183	0.000433	0.001017

WRITE

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
40000	-	10000.0	5000.0	5000.0	2500.0	1081.1
80000	-	13333.3	10000.0	5714.3	2000.0	1000.0

READ

ΧΡΟΝΟ ΑΠΟΚΡΙΣΗΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
60000	-	12000.0	5454.5	5000.0	2500.0	1132.1
120000	-	13333.3	10000.0	5454.5	2307.7	983.6

WRITE

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
40000	-	2.000	8.000	8.000	16.000	37.000
80000	-	6.000	8.000	14.000	40.000	80.000

READ

ΣΥΝΟΛΙΚΟ ΚΟΣΤΟΣ	ΝΗΜΑΤΑ	5	10	20	50	100
ΛΕΙΤΟΥΡΓΙΕΣ	-	-	-	-	-	-
60000	-	2.000	11.000	12.000	24.000	53.000
120000	-	9.000	12.000	22.000	52.000	122.000

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make clean
cd engine && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
rm -rf *.o libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make clean
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
rm -f kiwi-bench
rm -rf testdb
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$ make all
cd engine && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'
    CC db.o
    CC memtable.o
    CC indexer.o
    CC sst.o
    CC sst_builder.o
    CC sst_loader.o
    CC sst_block_builder.o
    CC hash.o
    CC bloom_builder.o
    CC merger.o
    CC compaction.o
    CC skiplist.o
    CC buffer.o
    CC arena.o
    CC utils.o
    CC crc32.o
    CC file.o
    CC heap.o
    CC vector.o
    CC log.o
    CC lru.o
    AR libindexer.a
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'
cd bench && make all
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'
gcc -g -ggdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -lpthread -lsnappy -o kiwi-bench
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'
myy601@myy601lab1:~/kiwi/kiwi-source$ cd bench/
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 100000 5
SELECTED ACTION: __WRITE__
+-----------------------------+----------------+------------------------------+-------------------+
|Random-WRITE    (done:100000, found:100000): 0.000050 sec/op; 20000.0 writes per sec(estimated); cost:5.000(sec)
```

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 100000 5
SELECTED ACTION:__READ__
+-----------------------------+----------------+------------------------------+-------------------+
|Random-READ    (done:100000, found:100000): 0.000050 sec/op; 20000.0 reads per sec(estimated); cost:5.000(sec)
```

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 100000 5
Keys:           16 bytes each
Values:         1000 bytes each
Entries:        100000
IndexSize:      2.4 MB (estimated)
DataSize:       95.7 MB (estimated)
------------------------------------------------------------------------------------------------
Date:           Mon Apr  3 17:19:51 2023
CPU:            1 *  Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz
CPUCache:       12288 KB

If you want the 50 percect of put and 50 percent of get choose 1.If you want the 90 percent put and 10 percent of get choose 2.If you want the 40 percent put and the 60 percent of get choose
3.
```