

# 1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

*Υλοποίηση πολυνηματικής λειτουργίας σε μηχανή αποθήκευσης  
δεδομένων*

Μέλη της ομάδας:

**Νικολάου Αλέξανδρος AM:4126**

**Κωνσταντίνος Μουρουσίδης AM:4114**

Σκοπος της ασκησης ηταν να υλοποιησουμε μια αποδοτικη πολυνηματικη λειτουργια των εντολων put και get που παρεχει η μηχανη αποθηκευσης δεδομενων.Η υλοποιηση μας εκτελει ταυτοχρονες λειτουργιες σωστα και διατηρει στατιστικα του χρονου εκτελεσης καθε λειτουργιας.Επισης διατηρει σε συνεπη μορφη την αποθηκη δεδομενων.Οι λειτουργιες που υλοποιηθηκαν ωστε να εκτελουνται πολυνηματικα ειναι η write,read και write\_read,που θα εξηγηθουν στις παρακατω ενοτητες.

## Ενότητα 1η

*Επεξήγηση των βασικών λειτουργιών(bench.c,bench.h)*

Η εκτέλεση ξεκινά με την κλήση της main μεθόδου του προγράμματος ως εξής:

Απο την γραμμή εντολών ο χρήστης θα πρέπει να διαλέξει την λειτουργία που θέλει με τους παρακάτω τρόπους.

1.

```
~/kiwi/kiwi-source/bench$ ./kiwi-bench write 100 2
```

Ο χρήστης επιλέγει να εκτελέσει την λειτουργία write για 100 εγγραφές(δεύτερο ορισμα) με 2 νηματα(τρίτο ορισμα).

2.

```
~/kiwi/kiwi-source/bench$ ./kiwi-bench read 100 2
```

Ο χρήστης επιλέγει να εκτελέσει την λειτουργία read για 100 αναγνώσεις(δεύτερο ορισμα) με 2 νηματα(τρίτο ορισμα).

3.

```
~/kiwi/kiwi-source/bench$ ./kiwi-bench write_read 100 2
```

```
select a percentage for writers and readers(50 50,70 30,40 60, etc.)  
50 50
```

Ο χρήστης επιλέγει να εκτελέσει την λειτουργία write\_read για 100 εγγραφές ή αναγνώσεις(δεύτερο ορισμα) με 2 νηματα(τρίτο ορισμα)

Και στη συνέχεια του ζητείτε το ποσοστό διαμοίρασης των νημάτων στους γραφείς και τους αναγνώστες με το πρώτο να αντιστοιχεί στους γραφείς και το δεύτερο στους αναγνώστες.

Η επεξεργασία των ορισμάτων της γραμμής εντολών γίνεται στην main στο αρχείο bench.c

```
209 int main(int argc, char** argv)
210 {
211     long int count;
212     int threads;
213     //initialize costs and locks
214     cost_of_writers=0;
215     cost_of_readers=0;
216     pthread_mutex_init(&writer_lock, NULL);
217     pthread_mutex_init(&reader_lock, NULL);
218
219     int writers = 0;    //to select write in method _create_writers_or_readers
220     int readers = 1;   //to select read in method _create_writers_or_readers
221
222     srand(time(NULL));
223     if (argc < 4) {    ///+1
224         fprintf(stderr, "Usage: db-bench <write | read | write_read> <count> <threads>\n");
225         exit(1);
226     }
227
228     if (strcmp(argv[1], "write") == 0) {
229         int r = 0;
230
231         count = atoi(argv[2]);
232         _print_header(count);
233         _print_environment();
234         if (argc == 5) ///+1
235             r = 1;
236
237         //////////////////////////////////////
238         threads = atoi(argv[3]);           //take threads number from command line
239         _db_open();                       //open db to write
240         _create_writers_or_readers(count, r, threads, writers); //write
241         _db_close();                     //close db
242         _print_writers_costs(count);      //threads are finished so print the total cost of writers
243         //////////////////////////////////////
244     } else if (strcmp(argv[1], "read") == 0) {
245         int r = 0;
246
247         count = atoi(argv[2]);
248         _print_header(count);
249         _print_environment();
250         if (argc == 5) ///+1
251             r = 1;
252
253         //////////////////////////////////////
254         threads = atoi(argv[3]);           //take threads number from command line
255         _db_open();                       //open db to read
256         _create_writers_or_readers(count, r, threads, readers); //read
257         _db_close();                     //close db
258         _print_readers_costs(count);      //threads are finished so print the total cost of readers
259         //////////////////////////////////////
260     }
```

```

261 //erotima 5
262 } else if(strcmp(argv[1], "write_read")==0){
263
264     int r = 0;
265     int writer_percentage;
266     int reader_percentage;
267     long int writer_count;
268     long int reader_count;
269     int array[2];
270     int* percentages;
271
272     count = atoi(argv[2]);
273     _print_header(count);
274     _print_environment();
275     if (argc == 5) ///+1
276         r = 1;
277
278     threads = atoi(argv[3]); //take threads number from command line
279
280     percentages = scan_percentages(array); //take percentages from user
281     writer_percentage = percentages[0];
282     reader_percentage = percentages[1];
283     writer_count = calculate_count(count,writer_percentage); //to know the new count
284     reader_count = calculate_count(count,reader_percentage); //to know the new count
285
286     _db_open(); //open db to write or read
287     _create_writers_and_readers(count,r,threads,writer_percentage,reader_percentage); //write and read
288     _db_close(); //close db
289     _print_writers_costs(writer_count); //threads are finished so print the total cost of writers
290     _print_readers_costs(reader_count); //threads are finished so print the total cost of readers
291 } else {
292     fprintf(stderr,"Usage: db-bench <write | read | write_read> <count> <threads>\n");
293     exit(1);
294 }
295
296 return 1;
297 }
298

```

## Υποενότητα 1.1

Αναλογα με το δευτερο ορισμα(read,write,write\_read) του χρηστη απο την γραμμη ενοτλων ικανοποιειται η καταλληλη συνθηκη.Στην πρωτη συνθηκη(write) οριζεται σε μια μεταβλητη threads ο απαιτουμενος αριθμος νηματων που εδωσε ο χρηστης.Στην συνεχεια ανοιγουμε την αποθηκη δεδομενων με μια βοηθητικη συναρτηση db\_open() και καλειται η συναρτηση create\_writers\_or\_readers().Τα ορισματα της συναρτησης ειναι πρωτο το count που τον αριθμο των εγγραφων,δευτερο το r,τριο ο αριθμος νηματων threads και τεταρτο μια βοηθητικη μεταβλητη που ικανοποιει την συνθηκη δημιουργιας writers.

Μετα την επιστροφη της συναρτησης create\_writers\_or\_readers()

Που σηματοδοτεί και το τέλος της λειτουργίας καλείται η συνάρτηση `db_close()` που κλείνει την αποθήκη δεδομένων. Και τέλος καλείται η συνάρτηση `print_writers_costs()` που δέχεται σαν ορίσμα το `count` (αριθμός εγγράφων) και εκτυπώνει τα κόστος των writers.

```
11 //helper method to open db      17 //helper method to close db
12 void _db_open()                 18 void _db_close()
13 {                               19 {
14     db = db_open(DATAS);        20     db_close(db);
15 }
```

Οι βοηθητικές συνάρτησεις που βοήθησαν στο άνοιγμα και το κλείσιμο της αποθήκης. Το άνοιγμα θα πρέπει να γίνεται πριν την δημιουργία νημάτων και το κλείσιμο μετά τον τερματισμό τους.

```
26 //create threads to write or read
27 void _create_writers_or_readers(long count,int r,int threads,int select)
28 {
29     struct arg_struct thread_args;
30     pthread_t some_thread[threads];
31     int i;
32
33     //initialize thread arguments
34     thread_args.count=count;
35     thread_args.r=r;
36     thread_args.threads=threads;
37     if(select==0) //create writers
38     {
39         //create threads
40         for(i=0;i<threads;i++)
41         {
42             pthread_create(&some_thread[i],NULL,call_write,(void *) &thread_args);
43         }
44         //wait thread to end
45         for(i=0;i<threads;i++)
46         {
47             pthread_join(some_thread[i],NULL);
48         }
49     }
50     if(select==1) //create readers
51     {
52         //create threads
53         for(i=0;i<threads;i++)
54         {
55             pthread_create(&some_thread[i],NULL,call_read,(void *) &thread_args);
56         }
57         //wait thread to end
58         for(i=0;i<threads;i++)
59         {
60             pthread_join(some_thread[i],NULL);
61         }
62     }
63 }
64 }
```

Πρωτα γινεται η αρχικοποιηση μιας δομης ορισματος(arg\_struct,γρ.29) η οποια μας βοηθαει να περασουμε τα ορισματα σε καθε νημα και αρχικοποιειται και ο πινακας με τα αναγνωριστικα των νημάτων.Επειτα καθοριζουμε τα ορισματα της δομης.Στην περιπτωση αυτη ακολουθειται η συνθηκη select==0 που δημιουργει νηματα για τους γραφεις,αυτο επιτυγχανεται με την κληση της pthread\_create δινοντας το αναγνωριστικο του νηματος,την συναρτηση που θελουμε να καλεσει call\_write και την δομη με τα ορισματα.Στην συνεχεια καλειται η pthread\_join που αναμενει τον τερματισμο των νημάτων.

```
5  //kanw ena struct gia na pernw ta orismata otan kanw pthread_create
6  struct arg_struct{
7      long int count;
8      int r;
9      int threads;
10 };
11
12 //helper function for pthread_create to call _write_test method
13 void *call_write(void *arg){
14     struct arg_struct *s = (struct arg_struct *) arg;
15     _write_test(s->count, s->r, s->threads);
16     return 0;
17 }
18
```

Η δομη περιχει τα τρια ορισματα που χρειαζομαστε για την εκτελεση της write\_test.

Η call\_write ειναι βοηθητικη συναρτηση που καλειται απο καθε νημα και η ιδια καλει την write\_test που χρειαζομαστε.

```
23 //helper method to print total cost of writers
24 void _print_writers_costs(long int count)
25 {
26     printf(LINE);
27     printf("|Random-Write   (done:%ld): %.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec);\n",
28         ,count, (double)(cost_of_writers / count)
29         ,(double)(count / cost_of_writers)
30         ,cost_of_writers);
31 }
32
```

Αφου τερματισουν ολα τα νηματα και κλεισουμε την αποθηκη δεδομενων καλουμε την `print_writers_costs`.Η οποια εκτυπωνει τα στατιστικα της αποδοσης της λειτουργιας(ερωτημα vi.).Τα στατιστικα προκυπτουν απο το συνολικο κοστος(δηλαδη το αθροισμα) των νηματων που κανουν τις εγγραφες.

## Υποενοτητα 1.2

Στην δευτερη συνθηκη(read) οριζεται σε μια μεταβλητη `threads` ο απαιτουμενος αριθμος νηματων που εδωσε ο χρηστης.Στην συνεχεια ανοιγουμε την αποθηκη δεδομενων με μια βοηθητικη συναρτηση `db_open()` και καλειται η συναρτηση `create_writers_or_readers()`.Τα ορισματα της συναρτησης ειναι πρωτο το `count` που τον αριθμο των αναγνωσεων,δευτερο το `r`,τριτο ο αριθμος νηματων `threads` και τεταρτο μια βοηθητικη μεταβλητη που ικανοποιει την συνθηκη δημιουργιας `readers`.

Μετα την επιστροφη της συναρτησης `create_writers_or_readers()`

Που σηματοδοτει και το τελος της λειτουργιας καλειται η συναρτηση `db_close()` που κλεινει την αποθηκη δεδομενων.Και τελος καλειται η συναρτηση `print_readers_costs()` που δεχεται σαν ορισμα το `count`(αριθμος αναγνωσεων) και εκτυπωνει τα κοστοι των `readers`.

Με διαφορα απο την `write` στην συναρτηση `create_writers_or_readers()` εδω ικανοποιειται η συνθηκη `select==1` που δημιουργει νηματα για τους αναγνωστες,αυτο επιτυγχανεται με την κληση της `pthread_create` δινοντας το αναγνωριστικο του νηματος,την συναρτηση που θελουμε να καλεσει `call_read` και την δομη με τα ορισματα.Στην συνεχεια καλειται η `pthread_join` που αναμενει τον τερματισμο των νηματων.

```

18
19 //helper function for pthread_create to call _read_test method
20 void *call_read(void *arg){
21     struct arg_struct *s = (struct arg_struct *) arg;
22     _read_test(s->count, s->r, s->threads);
23     return 0;
24 }

```

Στην περίπτωση αυτή τα νημάτα καλούν την βοηθητική συνάρτηση `call_read` και η ίδια καλεί την `read_test` που χρειαζόμαστε.

```

33 //helper method to print total cost of readers
34 void _print_readers_costs(long int count)
35 {
36     printf(LINE);
37     printf("|Random-Read    (done:%ld): %.6f sec/op; %.1f reads/sec(estimated); cost:%.3f(sec);\n"
38           ,count, (double)(cost_of_readers / count)
39           ,(double)(count / cost_of_readers)
40           ,cost_of_readers);
41 }
42 ///////////////////////////////////////////////////
43

```

Αφού τερματίσουν όλα τα νημάτα και κλείσουμε την αποθήκη δεδομένων καλούμε την `print_readers_costs`. Η οποία εκτυπώνει τα στατιστικά της απόδοσης της λειτουργίας (ερώτημα vi.). Τα στατιστικά προκύπτουν από το συνολικό κόστος (δηλαδή το άθροισμα) των νημάτων που κάνουν τις αναγνώσεις.

### Υποενοτητα 1.3

Στην τρίτη συνθήκη (`write_read`) ορίζεται σε μια μεταβλητή `threads` ο απαιτούμενος αριθμός νημάτων που έδωσε ο χρήστης. Σε μια μεταβλητή `percentages` επιστρέφεται από την κλήση της συνάρτησης `scan_percentages()` τα ποσοστά που έδωσε ο χρήστης σε μορφή



πινακα.Με τα ποσοστα και το αριθμο count υπολογιζουμε με την συναρτηση calculate\_cou.Στην συνεχεια ανοιγουμε την αποθηκη δεδομενων με μια βοηθητικη συναρτηση db\_open() και καλειται η συναρτηση create\_writers\_and\_readers().Τα ορισματα της συναρτησης ειναι πρωτο το count που τον αριθμο των αναγνωσεων,δευτερο το r,τριτο ο αριθμος νηματων threads,τεταρτο το ποσοστο για τους γραφεις και πεμπτο το ποσοστο για τους αναγνωστες.

```
65 //create thread to write and read
66 void _create_writers_and_readers(long int count,int r,int threads,int writer_percentage,int reader_percentage)
67 {
68     struct arg_struct writer_args;
69     struct arg_struct reader_args;
70     pthread_t writer[threads];
71     pthread_t reader[threads];
72     int i;
73
74     //initialize writer thread arguments
75     writer_args.count=(long) (count*writer_percentage/100);
76     writer_args.r=r;
77     writer_args.threads=(int) (threads*writer_percentage/100);
78
79     //initialize reader thread arguments
80     reader_args.count=(long) (count*reader_percentage/100);
81     reader_args.r=r;
82     reader_args.threads=(int) (threads *reader_percentage/100);
83
84     //create writer thread
85     for(i=0;i<writer_args.threads;i++)
86     {
87         pthread_create(&writer[i],NULL,call_write,(void *) &writer_args);
88     }
89     //create reader thread
90     for(i=0;i<reader_args.threads;i++)
91     {
92         pthread_create(&reader[i],NULL,call_read,(void *) &reader_args);
93     }
94     //wait readers to end
95     for(i=0;i<reader_args.threads;i++)
96     {
97         pthread_join(reader[i],NULL);
98     }
99     ///wait writers to end
100     for(i=0;i<writer_args.threads;i++)
101     {
102         pthread_join(writer[i],NULL);
103     }
104 }
```

Αρχικοποιουνται δυο δομες ορισματων(μια για την κληση της write\_test και μια για την κληση της read\_test).Επισης αρχικποιουνται δυο πινακες αναγνωριστικων των νηματων.Υπολογιζουμε το νεο count που εχουμε και το νεο αριθμο νηματων με βαση τα ποσοστα και καθοριζουμε καθε ορισμα των δομων μας.Επειτα δημιουργει νηματα για τους γραφεις και τους αναγνωστες,αυτο επιτυγχανεται με την

κλήση της pthread\_create δίνοντας το αναγνωριστικό του νηματος,την συναρτηση που θελουμε να καλεσει call\_write ή call\_read και την δομη με τα ορισματα.Στην συνεχεια καλειται η pthread\_join που αναμενει τον τερματισμο των νηματων.

```
106 //scan the percentages from user
107 int* scan_percentages(int *array)
108 {
109     char percentages[5];
110     char space[]=" ";
111     int writer_percentage;
112     int reader_percentage;
113
114     printf("select a percentage for writers and readers(50 50,70 30,40 60, etc.)\n");
115     scanf("%[^\\n]s",percentages);
116     //check if the user write something wrong
117     while(isdigit(percentages[0])==0 || isdigit(percentages[1])==0 || isspace(percentages[2])==0
118           || isdigit(percentages[3])==0 || isdigit(percentages[4])==0 || strlen(percentages)>5)
119     {
120         printf("Wrong percentage please try again(50 50,70 30,40 60, etc.)\n");
121         getchar();
122         scanf("%[^\\n]s",percentages);
123     }
124
125     char *ptr = strtok(percentages, space);
126     writer_percentage =atoi(ptr);
127     ptr = strtok(NULL, space);
128     reader_percentage =atoi(ptr);
129     array[0]=writer_percentage;
130     array[1]=reader_percentage;
131     return array; //return the percentages
132 }
```

Η scan\_percentages() ζηται απο τον χρηστη να καθορισει το ποσοστο ταυτοχρονισμού οταν εκτελειται η λειτουργια write\_read.Και επιστρεφει ενα πινακα με δυο ποσοστα.

```
133 //return count multiplied by the percentage
134 long int calculate_count(long int count,int percentage)
135 {
136     long int percentage_count;
137     percentage_count = (long) (count*percentage/100);
138     return percentage_count;
139 }
```

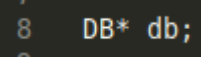
Η calculate\_count παρνει σαν ορισματα ενα count και ενα ποσοστο και επιστρεφει το count πολλαπλασιασμενο με το ποσοστο.

Οταν τερματισουν ολα τα νηματα(γραφεις και αναγνωστες) κλεινουμε την αποθηκη δεδομενων και εκτυπωνουμε τα στατιστικα απο τους αναγνωστες και τους γραφεις.

Τελος,στο αρχειο bench.h οριζουμε τα συνολικα κοστη των γραφειων και των αναγνωστων και δυο κλειδαριες που θα εξηγησουμε παρακατω την χρηση τους.

## Ενότητα 2η

*Επεξηγηση των παρεμβασεων στο αρχειο kiwi.c*

Αρχικα αφαιρεθηκε το db\_open(),db\_close() απο τις συναρτησεις write\_test,read\_test ωστε να αποφευχθει το μη επιθυμητο ανοιγμα και κλεισιμο απο καθε νημα.Αντιθετως γινεται ενα ανοιγμα πριν την δημιουργια νηματων και ενα κλεισιμο μετα τον τερματισμο τους,στις συναρτησεις που εξηγησαμε παραπανω.Για τον λογο αυτο το εγινε global το . `DB* db;`

Στις write\_test και read\_test προστεθηκε ακομη ενα οριμα που αναφερει το αριθμο των νηματων που θελουμε.

```

44 void _write_test(long int count, int r,int threads) //add an argument to know threads number
45 {
46     int i;
47     double cost;
48     long long start,end;
49     Variant sk, sv;
50     long int threads_count; //new count
51
52     char key[KSIZE + 1];
53     char val[VSIZE + 1];
54     char sbuf[1024];
55
56     memset(key, 0, KSIZE + 1);
57     memset(val, 0, VSIZE + 1);
58     memset(sbuf, 0, 1024);
59
60     start = get_ustime_sec();
61
62     threads_count = count/threads; //split the work for every thread
63     for (i = 0; i < threads_count; i++) {
64         if (r)
65             _random_key(key, KSIZE);
66         else
67             snprintf(key, KSIZE, "key-%d", i);
68         fprintf(stderr, "%d adding %s\n", i, key);
69         snprintf(val, VSIZE, "val-%d", i);
70
71         sk.length = KSIZE;
72         sk.mem = key;
73         sv.length = VSIZE;
74         sv.mem = val;
75
76         db_add_helper(db, &sk, &sv); //call a helper method to add
77         if ((i % 10000) == 0) {
78             fprintf(stderr, "random write finished %d ops%30s\r",
79                     i,
80                     "");
81
82             fflush(stderr);
83         }
84     }
85     end = get_ustime_sec();
86     cost = end - start;
87     pthread_mutex_lock(&writer_lock); //lock from other threads
88     cost_of_writers=cost_of_writers+cost; //add on total cost the cost of this thread
89     pthread_mutex_unlock(&writer_lock); //unlock
90 }
91

```

Πρωτη δουλεια ειναι να υπολογισουμε το νεο count που θα προκυπτει απο την διαιρεση του count με τον αριθμο των νηματων ωστε να εξασφαλισουμε να εχουμε ισες εγγραφες απο καθε νημα.

Επισης αντι για την κληση της db\_add καλουμε μια βοηθητικη συναρτηση που θα εξηγησουμε παρακατω.Λογω το οτι εχουμε πολλα νηματα που εκτελουνται για να υπολογισουμε το σωστο κοστος θα πρεπει απο καθε νημα να προσθεσουμε το κοστος του σε μια μεταβλητη που κραταει το συνολικο(cost\_of\_writers).Πριν γινει αυτη η πραξη θα πρεπει να κλειδωσουμε την κρισιμη περιοχη για να μην

παρεμβάλει άλλο νήμα. Και έπειτα να ξεκλειδώσουμε την κρίσιμη περιοχή.

```
92 void _read_test(long int count, int r, int threads)
93 {
94     int i;
95     int ret;
96     int found = 0;
97     double cost;
98     long long start, end;
99     Variant sk;
100     Variant sv;
101
102     char key[KSIZE + 1];
103
104     long int threads_count;           //new count
105
106     start = get_ustime_sec();
107     threads_count = count/threads;    //split the work for every thread
108     for (i = 0; i < threads_count; i++) {
109         memset(key, 0, KSIZE + 1);
110
111         /* if you want to test random write, use the following */
112         if (r)
113             _random_key(key, KSIZE);
114         else
115             snprintf(key, KSIZE, "key-%d", i);
116         fprintf(stderr, "%d searching %s\n", i, key);
117         sk.length = KSIZE;
118         sk.mem = key;
119         ret = db_get_helper(db, &sk, &sv);    //call a helper method to get
120         if (ret) {
121             //db_free_data(sv.mem);
122             found++;
123         } else {
124             INFO("not found key#%s",
125                 sk.mem);
126         }
127
128         if ((i % 10000) == 0) {
129             fprintf(stderr, "random read finished %d ops%30s\r",
130                 i,
131                 "");
132
133             fflush(stderr);
134         }
135     }
136     end = get_ustime_sec();
137     cost = end - start;
138     pthread_mutex_lock(&reader_lock);        //lock from other threads
139     cost_of_readers = cost_of_readers + cost; //add on total cost the cost of this thread
140     pthread_mutex_unlock(&reader_lock);      //unlock
141 }
142
```

Πρώτη δουλειά είναι να υπολογίσουμε το νέο count που θα προκύπτει από την διαίρεση του count με τον αριθμό των νημάτων ώστε να εξασφαλίσουμε να έχουμε ίσες αναγνώσεις από κάθε νήμα.

Επίσης αντί για την κλήση της db\_get καλούμε μια βοηθητική συνάρτηση που θα εξηγήσουμε παρακάτω. Λόγω το ότι έχουμε πολλά νήματα που εκτελούνται για να υπολογίσουμε το σωστό κόστος θα πρέπει από κάθε νήμα να προσθέσουμε το κόστος του σε μια

μεταβλητη που κραταει το συνολικο(cost\_of\_readers).Πριν γίνει αυτη η πραξη θα πρεπει να κλειδωσουμε την κρισιμη περιοχη για να μην παρεμβάλει άλλο νημα.Και επειτα να ξεκλειδωσουμε την κρισιμη περιοχη.

Οι κλειδαριες αυτες οπως αναφεραμε νωριτερα οριζονται στο bench.h και αρχικοποιουνται μεσα στην main στο bench.c.

## Ενότητα 3η

*Υποστηριξη ταυτοχρονων λειτουργιων add και get(db.c,db.h)*

Με σκοπο την διατηρηση της συνεπειας της αποθηκης δεδομενων επιτρεπουμε την εισοδο ενος γραφει ή πολλαπων αναγνωστων ταυτοχρονα.Αυτο επιτυγχανεται με την χρηση καταλληλων κλειδαριων στις κρισιμες περιοχες.

```
11
12  typedef struct _db {
13      // char basedir[MAX_FILENAME];
14      char basedir[MAX_FILENAME+1];
15      SST* sst;
16      MemTable* memtable;
17      pthread_mutex_t general_lock;    //lock for writers and readers
18      pthread_mutex_t readers_lock;   //lock for readers
19      int readers;                     //helper variable for how many readers are in
20  } DB;
21
```

Αρχικα προσθετουμε τρια νεα πεδια στο struct \_db:δυο κλειδαριες και μια βοηθητικη μεταβλητη.

Αυτα αρχικοποιουνται μεσα στο db.c στην συναρτηση db\_open\_ex ωστε καθε φορα που θα ανοιγει η αποθηκη να δημιουργουνται νεες κλειδαριες και η βοηθητικη μεταβλητη να ισουται με το

μηδεν(γρ.23,24,25).

```
9
10 DB* db_open_ex(const char* basedir, uint64_t cache_size)
11 {
12     DB* self = calloc(1, sizeof(DB));
13
14     if (!self)
15         PANIC("NULL allocation");
16
17     strncpy(self->basedir, basedir, MAX_FILENAME);
18     self->sst = sst_new(basedir, cache_size);
19
20     Log* log = log_new(self->sst->basedir);
21     self->memtable = memtable_new(log);
22
23     pthread_mutex_init(&(self->general_lock), NULL);           //initialize general lock
24     pthread_mutex_init(&(self->readers_lock), NULL);          //initialize readers lock
25     self->readers=0;                                           //initialize variable
26
27     return self;
28 }
```

```
35
36 int db_add_helper(DB* self, Variant* key, Variant* value)
37 {
38     int var;
39     pthread_mutex_lock(&self->general_lock);                  //lock for other writers and all readers because we want to write
40     var = db_add(self, key, value);                            //call db_add and pass the returned variable (write)
41     pthread_mutex_unlock(&self->general_lock);                 //unlock
42     return var;
43 }
44
```

Η συνάρτηση αυτή όπως αναφεραμε προηγουμένως καλείται στην συνάρτηση write\_test() και εδώ θα εξηγήσουμε την λειτουργία της.Για να μπορέσουμε να εξασφαλίσουμε την εισοδο ενός μονο γραφεία χρησιμοποιείται η κλειδαριά general\_lock που ορίστηκε στο db.h.Στην κρισιμη περιοχή καλείται η db\_add

```
85
86 int db_add(DB* self, Variant* key, Variant* value)
87 {
88     if (memtable_needs_compaction(self->memtable))
89     {
90         INFO("Starting compaction of the memtable after %d insertions and %d deletions",
91             self->memtable->add_count, self->memtable->del_count);
92         sst_merge(self->sst, self->memtable);
93         memtable_reset(self->memtable);
94     }
95
96     return memtable_add(self->memtable, key, value);
97 }
```

Η οποία εκτελεί την εγγραφή στην αποθήκη.Αφου ολοκληρωθει η εγγραφή μπορούμε να ξεκλειδωσουμε την `general_lock`.Και τέλος να επιστρέψουμε την μεταβλητη που επιστρέφεται απο την `db_add` ώστε να συνεχιστει η εκτελεση του προγραμματος.

```
45 int db_get_helper(DB* self, Variant* key, Variant* value)
46 {
47     int var; //helper variable to return in the end of the method
48     pthread_mutex_lock(&self->readers_lock); //lock for other readers
49     self->readers++; //add a reader
50     if(self->readers==1)
51     {
52         pthread_mutex_lock(&self->general_lock); //lock for writers too
53     }
54     pthread_mutex_unlock(&self->readers_lock); //unlock for readers
55     var = db_get(self, key, value); //call db_get and pass the returned variable (read)
56     pthread_mutex_lock(&self->readers_lock); //lock for other readers
57     self->readers--; //remove a reader
58     if(self->readers==0)
59     {
60         pthread_mutex_unlock(&self->general_lock); //unlock for writers
61     }
62     pthread_mutex_unlock(&self->readers_lock); //unlock for readers
63
64     return var;
65 }
66
```

Η συναρτηση αυτη οπως αναφεραμε προηγουμενως καλειται στην συναρτηση `read_test()` και εδω θα εξηγησουμε την λειτουργια της.Για να μπορεσουμε να εξασφαλισουμε την εισοδο πολλων αναγνωστων ταυτοχρονα θα πρεπει να χρησιμοποιησουμε δυο κλειδαριες.Η πρωτη κλειδαρια ειναι η `reader_lock` ώστε οταν μπουν οι αναγνωστες να κλειδωσουν την κρισιμη περιοχη που γινεται η αυξηση κατα ενα της μεταβλητης μας `readers`.Στην ουσια αυτο γινεται για να ξερουμε ποσοι αναγνωστες βρισκονται μεσα στην αποθηκη.Ο πρωτος αναγνωστης θα πρεπει να κλειδωσει την δευτερη κλειδαρια(`general_lock`) ώστε να αποτρεψει εναν γραφει να γραψει στην αποθηκη την στιγμη της αναγνωσης.Εφοσον αυξηθηκε η `readers` κατα ενα ξεκλειδωνουμε την `readers_lock` και καλουμε την



db\_get.Επειτα οταν ενας αναγνωστης τελειωσει την αναγνωση θα πρεπει να κλειδωσει την κλειδαρια readers\_lock για να μειωσει κατα ενα την readers(δηλαδη να δηλωση την αποχωρηση του) και να ξεκλειδωσει την κλειδαρια readers\_lock.Αν ο αναγνωστης που θελει να αποχωρησει ειναι ο τελευταιος στην αποθηκη θα ξεκλειδωσει και την κλειδαρια general\_lock ωστε να μپορει να μπει και καποιος γραφεας.Με αυτον τον τροπο επιτυγχανεται η διατηρηση της συνεπειας της αποθηκης δεδομενων.

## Ενότητα 4η

*Παρουσιαση στατιστικων αποδοσης*

Τα στατικα εκτυπωνονται απο τις συναρτησεις \_print\_writers\_costs και \_print\_readers\_costs οπου για να βρεθει το κοστος ανα λειτουργια το κοστος διαιρειται με τον αριθμο νηματων του χρησιμοποιηθηκαν.

- *Εκτελεση της make*

```

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$

```

- Εκτέλεση της λειτουργίας write

1.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 1000000 10

```

```

[1207] 31 Mar 2016:10:10:00 : log file removing old log file test00/31/242.log
+-----+
|Random-Write  (done:1000000): 0.000090 sec/op; 11111.1 writes/sec(estimated); cost:90.000(sec);

```

2.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 1000000 30

```

```

[1300] 31 Mar 2016:10:10:10 : log file removing old log file test00/31/242.log
+-----+
|Random-Write  (done:1000000): 0.000305 sec/op; 3278.7 writes/sec(estimated); cost:305.000(sec);
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

3.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 1000000 60

```

```

[1412] 31 Mar 2016:10:10:20 : log file removing old log file test00/31/242.log
+-----+
|Random-Write  (done:1000000): 0.000603 sec/op; 1658.4 writes/sec(estimated); cost:603.000(sec);
myy601@myy601lab1:~/kiwi/kiwi-source/bench$

```

4.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 1000000 100
```

```
[2104] 31 Mar 2016 11:51:17 + log: file removing old log file testab/31/2121.log
+-----+
|Random-Write   (done:1000000): 0.000945 sec/op; 1058.2 writes/sec(estimated); cost:945.000(sec);
```

- Εκτέλεση της λειτουργίας read

1.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 1000000 10
```

```
+-----+
|Random-Read    (done:1000000): 0.000040 sec/op; 25000.0 reads/sec(estimated); cost:40.000(s
```

2.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 1000000 30
```

```
+-----+
|Random-Read    (done:1000000): 0.000090 sec/op; 11111.1 reads/sec(estimated); cost:90.000(s
```

3.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 1000000 60
```

```
[2200] 31 Mar 2016 11:51:17 + skip:1000000/ skip:1000000 is at 0.1 freeing up the structure
+-----+
|Random-Read    (done:1000000): 0.000180 sec/op; 5555.6 reads/sec(estimated); cost:180.000(sec);
```

4.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 1000000 100
```

```
+-----+
|Random-Read    (done:1000000): 0.000300 sec/op; 3333.3 reads/sec(estimated); cost:300.000(sec);
```

- Εκτέλεση της λειτουργίας write\_read

1.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write_read 1000000 10
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1000000
IndexSize:     23.8 MB (estimated)
DataSize:      957.5 MB (estimated)
-----
Date:          Thu Mar 31 20:35:07 2022
CPU:           1 * AMD Ryzen 5 PRO 4650G with Radeon Graphics
CpuCache:
select a percentage for writers and readers(50 50,70 30,40 60, etc.)
60 40
+-----+-----+-----+-----+-----+-----+
|Random-Write  (done:600000): 0.000100 sec/op; 10000.0 writes/sec(estimated); cost:60.000(sec);
+-----+-----+-----+-----+-----+-----+
|Random-Read   (done:400000): 0.000090 sec/op; 11111.1 reads/sec(estimated); cost:36.000(sec);
+-----+-----+-----+-----+-----+-----+

```

2.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write_read 1000000 30
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1000000
IndexSize:     23.8 MB (estimated)
DataSize:      957.5 MB (estimated)
-----
Date:          Thu Mar 31 20:36:52 2022
CPU:           1 * AMD Ryzen 5 PRO 4650G with Radeon Graphics
CpuCache:
select a percentage for writers and readers(50 50,70 30,40 60, etc.)
50 50
+-----+-----+-----+-----+-----+-----+
|Random-Write  (done:500000): 0.000288 sec/op; 3472.2 writes/sec(estimated); cost:144.000(sec);
+-----+-----+-----+-----+-----+-----+
|Random-Read   (done:500000): 0.000240 sec/op; 4166.7 reads/sec(estimated); cost:120.000(sec);
+-----+-----+-----+-----+-----+-----+

```

3.

```

myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write_read 1000000 60
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1000000
IndexSize:     23.8 MB (estimated)
DataSize:      957.5 MB (estimated)
-----
Date:          Thu Mar 31 20:38:38 2022
CPU:           1 * AMD Ryzen 5 PRO 4650G with Radeon Graphics
CpuCache:
select a percentage for writers and readers(50 50,70 30,40 60, etc.)
40 60

```

```
+-----+-----+-----+-----+
|Random-Write  (done:400000): 0.000483 sec/op; 2072.5 writes/sec(estimated); cost:193.000(sec);
+-----+-----+-----+-----+
|Random-Read   (done:600000): 0.000412 sec/op; 2429.1 reads/sec(estimated); cost:247.000(sec);
+-----+-----+-----+-----+
```

4.

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write_read 1000000 100
Keys:          16 bytes each
Values:        1000 bytes each
Entries:       1000000
IndexSize:     23.8 MB (estimated)
DataSize:      957.5 MB (estimated)
-----
Date:          Thu Mar 31 20:40:14 2022
CPU:           1 * AMD Ryzen 5 PRO 4650G with Radeon Graphics
CPUCache:
select a percentage for writers and readers(50 50,70 30,40 60, etc.)
80 20

+-----+-----+-----+-----+
|Random-Write  (done:800000): 0.000923 sec/op; 1084.0 writes/sec(estimated); cost:738.000(sec);
+-----+-----+-----+-----+
|Random-Read   (done:200000): 0.000480 sec/op; 2083.3 reads/sec(estimated); cost:96.000(sec);
+-----+-----+-----+-----+
```

Παρατηρούμε ο,τι με την αύξηση των νημάτων, κάποιος θα περιμενε να μειωνεται το κοστος,αυτο ομως δε συμβαινει γιατι το κοστος καθε φορα που εκτελειται μια λειτουργια αυξανεται καθως προκειται για το συνολικο κοστος.Ομως το κοστος για καθε νημα ξεχωριστα ειναι μικροτερο αφου επι της ουσιας εχει να εκτελεσει λιγοτερες εγγραφες ή αναγνωσεις λογω του διαμοιρασμου του φορτου.

Επισης στην εκτελεση της λειτουργιας write\_read παρατηρούμε οτι μπορεί καποιες φορες ενας αναγνωστης να μην μπορεί να βρει καποιο κλειδι, αυτο ειναι λογικο καθως υπαρχει περιπτωση να μην εχει ο κατλληλος αριθμος εγγραφων στην αποθηκη δεδομενων.

Σπανια σε περιπτώσεις μεγαλου φορτου και πολλαπλων νηματων το συστημα δεν ανταποκρινεται και εμφανιζει το bloom offset.Ομως με επαναληψη της λειτουργιας εκτελειται κανονικα.