

“Υλοποίηση πολυνηματικής
λειτουργίας σε μηχανή αποθήκευσης δεδομένων”



Εργαστήριο Λειτουργικών Συστημάτων, 4 Απριλίου 2025

Δήμητρα Τζιάφα Α.Μ. : 5366

Ηλιάννα Ράππη Α.Μ. : 5128

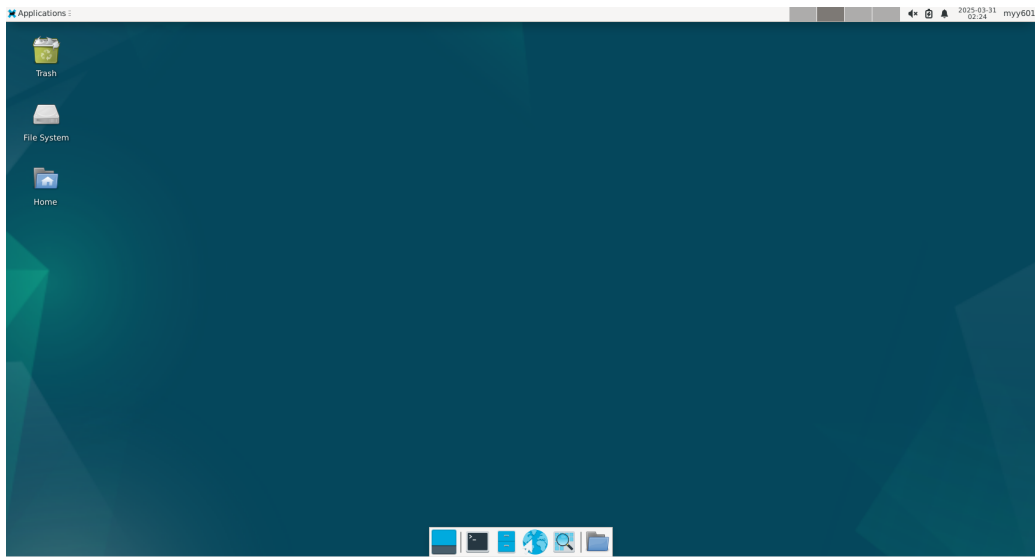
Περιεχόμενα

1	Εισαγωγή	3
1.1	Εγκατάσταση εικονικής μηχανής - Προεργασία αρχικού κώδικα . . .	3
1.2	Αρχικές μετρήσεις κόστους - <i>Casting</i> σε <i>double</i>	3
1.3	Αποτελέσματα αρχικών μετρήσεων κόστους	4
2	Επεξεργασία/Τροποποίηση κώδικα - Αμοιβαίος αποκλεισμός μεταξύ <i>add</i> και <i>get</i>, συγκρούσεις νημάτων μηχανής	6
2.1	Επεξεργασία αρχείου <i>db.c</i> - εφαρμογή αμοιβαίου αποκλεισμού . . .	6
2.2	Επεξεργασία αρχείου <i>db.c</i> - επίλυση προβλήματος σύγκρουσης μεταξύ νημάτων της μηχανής	7
3	Επεξεργασία/Τροποποίηση κώδικα - Προετοιμασία για προσθήκη νημάτων - Λειτουργία <i>readwrite</i>	8
3.1	Επεξεργασία αρχείου <i>kiwi.c</i> - τροποποίηση <i>_write_test()</i> , <i>_read_test()</i>	8
3.2	Επεξεργασία αρχείου <i>bench.c</i> - Προσαρμογή αλλαγών από το αρχείο <i>kiwi.c</i>	13
3.3	Επεξεργασία αρχείου <i>bench.c</i> - Βελτιστοποίηση λειτουργιών <i>write</i> , <i>read</i> , προσθήκη λειτουργίας <i>readwrite</i>	13
4	Έξοδος τελικής εντολής <i>make all</i>	16
5	Πληροφορίες συστήματος	17

1 Εισαγωγή

1.1 Εγκατάσταση εικονικής μηχανής - Προεργασία αρχικού κώδικα

Σε πρώτη φάση, εγκαταστάθηκε ο *VMware Workstation Pro v17.6.2* και ύστερα τοποθετήθηκε η εικονική μηχανή, ακολουθώντας τα βήματα που δίνονται στο *pdf* της εκφώνησης. Καθ'όλη τη διάρκεια της προετοιμασίας, δεν προέκυψε κάποιο πρόβλημα στο *set up* του *VM*. Έχοντας εκτελέσει σωστά τα βήματα της εγκατάστασης, πραγματοποιείται σύνδεση στο *VM* όπου και εμφανίζεται το περιβάλλον της εικονικής μηχανής ως εξής:



Σε δεύτερη φάση, ακολούθησε περιεργασία του υπάρχοντος κώδικα με σκοπό την κατανόηση των λειτουργιών *add* και *get* που επιτελεί η μηχανή αποθήκευσης, τη διερεύνηση των νημάτων που υφίστανται ήδη στην υλοποίηση που δίνεται, καθώς και τον ρόλο των βασικών μεθόδων και της μεταξύ τους συνεργασίας.

1.2 Αρχικές μετρήσεις κόστους - *Casting σε double*

Με σκοπό την περαιτέρω κατανόηση των παραπάνω, πραγματοποιήθηκαν μερικές πειραματικές εγγραφές/αναγνώσεις με διαφορετικό πλήθος στοιχείων κάθε φορά. Ωστόσο, κατά τη διάρκεια των εκτελέσεων, διαπιστώθηκε ότι τα κόσθη έβγαιναν σε όλες τις περιπτώσεις μηδενικά. Ως αποτέλεσμα αυτού, πραγματοποιήθηκε το εξής *type casting* στις συναρτήσεις *_read_test()*, *_write_test()* του αρχείου *kiwi.c* με σκοπό την αποτύπωση αντικειμενικών στατιστικών απόδοσης:

```

Για Write :
printf("|Random-Read      (done:%ld, found:%d): %.6f sec/op; %.1f reads /sec(estimated); cost:%.3f(sec)\n",
      count, found,
      cost / (double) count,
      count / (double) cost,
      cost);

Για Read :
printf("|Random-Read      (done:%ld, found:%d): %.6f sec/op; %.1f reads /sec(estimated); cost:%.3f(sec)\n",
      count, found,
      cost / (double) count,
      count / (double) cost,
      cost);

```

1.3 Αποτελέσματα αρχικών μετρήσεων κόστους

Στο path `~/kiwi/kiwi-source`, δημιουργήθηκαν τα εκτελέσιμα αρχεία με χρήση της εντολής `make all` και έπειτα με μετακίνηση στον κατάλογο `bench`, εκτελέστηκαν διάφορες εγγραφές/αναγνώσεις μέσω των εντολών `./kiwi - bench write x` και `./kiwi - bench read x` αντίστοιχα, όπου το `x` αναπαριστά το πλήθος των στοιχείων. Επιλέχθηκε ένα ευρύ πλήθος στοιχείων για δοκιμές, μεταξύ των οποίων ήταν οι: `x = 10.000, 50.000, 100.000, 500.000, 800.000, 1.000.000, 1.500.000`. Για τιμές μικρότερες του 10.000, θεωρήθηκε άσκοπο να συμπεριληφθούν στα τελικά αποτελέσματα καθώς, δεδομένου του μικρού αριθμού της εισόδου, η εκτέλεση τερμάτιζε γρήγορα και το κόστος ήταν ίσο με `inf`.

Παραδείγματα διάφορων εκτελέσεων που πραγματοποιήθηκαν στα πλαίσια του πειραματικού σταδίου:

- Εγγραφή και ανάγνωση 100.000 στοιχείων:

```

|Random-Write (done:100000): 0.000020 sec/op; 50000.0 writes/sec(estimated); cost:2.000(sec);
+-----+
|Random-Read  (done:100000, found:100000): 0.000020 sec/op; 50000.0 reads /sec(estimated); cost:2.000(sec)
+-----+

```

- Εγγραφή και ανάγνωση 500.000 στοιχείων:

```

|Random-Write (done:500000): 0.000018 sec/op; 55555.6 writes/sec(estimated); cost:9.000(sec);
+-----+
|Random-Read  (done:500000, found:500000): 0.000010 sec/op; 100000.0 reads /sec(estimated); cost:5.000(sec)
+-----+

```

- Εγγραφή και ανάγνωση 1.000.000 στοιχείων:

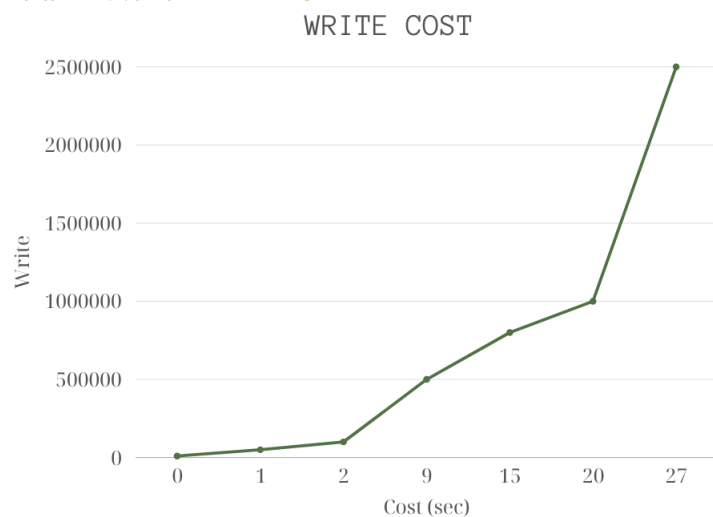
```

|Random-Write (done:1000000): 0.000020 sec/op; 50000.0 writes/sec(estimated); cost:20.000(sec);
+-----+
|Random-Read  (done:1000000, found:1000000): 0.000009 sec/op; 111111.1 reads /sec(estimated); cost:9.000(sec)
+-----+

```

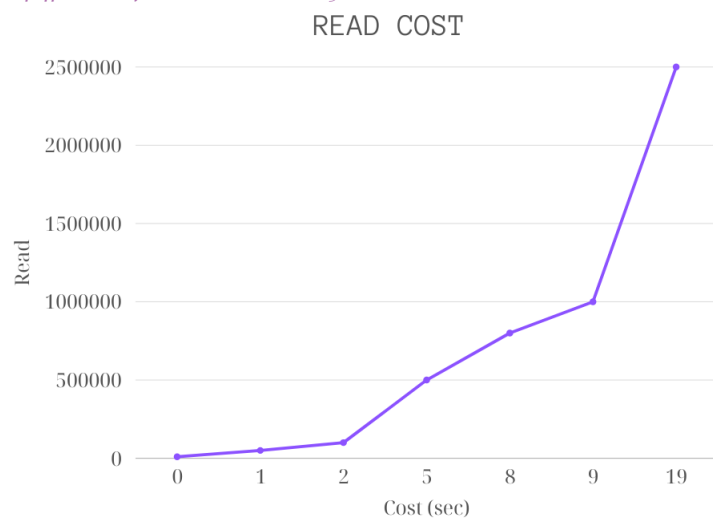
Για λόγους καλύτερης οπτικοποίησης, δημιουργήθηκαν δύο γραφήματα των αποτελεσμάτων που λήφθηκαν:

Γράφημα Εγγραφών-Κόστους



WRITE	10000	50000	100000	500000	800000	1000000	1500000
COST	0	1	2	9	15	20	27

Γράφημα Αναγνώσεων-Κόστους



READ	10000	50000	100000	500000	800000	1000000	1500000
COST	0	1	2	5	8	9	19

Όπως φαίνεται και παρακάτω, στο γράφημα Εγγραφών - Κόστους, παρατηρείται μια ξαφνική αύξηση του κόστους μεταξύ των 100000 και 50000 εγγραφών από 2 σε 9 δευτερόλεπτα. Αυτό, οφείλεται στο γεγονός πως εκείνη τη στιγμή πραγματοποιείται η διαδικασία του *compaction*, δηλαδή η σύμπτυξη των εγγραφών ενός επιπέδου i με αυτών του $i+1$ στο αρχείο *sst* και παράλληλα η ανακατανομή τους με σκοπό την ταξινόμησή τους. Ωστόσο, και στις δύο περιπτώσεις, το κόστος διεκπεραίωσης κάθε λειτουργίας μεγαλώνει, όσο αυξάνεται και το πλήθος των εγγραφών.

2 Επεξεργασία/Τροποποίηση κώδικα - Αμοιβαίος αποκλεισμός μεταξύ *add* και *get*, συγκρούσεις νημάτων μηχανής

2.1 Επεξεργασία αρχείου *db.c* - εφαρμογή αμοιβαίου αποκλεισμού

Στο στάδιο αυτό, πραγματοποιούνται σταδιακές αλλαγές στον υπάρχοντα κώδικα με σκοπό την εφαρμογή του αμοιβαίου αποκλεισμού ανάμεσα στις λειτουργίες εγγραφής και ανάγνωσης στη δομή ώστε να εξασφαλίζεται ότι μόνο μία λειτουργία κάθε φορά θα χρησιμοποιεί τη μηχανή αποθήκευσης. Αυτό, θα επιτευχθεί με τη χρήση κλειδαριών αμοιβαίου αποκλεισμού *mutex*, ως εξής:

Αρχικά, στο αρχείο *db.h* συμπεριλαμβάνεται η βιβλιοθήκη *pthread.h*:

```
#include <pthread.h>
```

ενώ, στο αρχείο *db.c*, δημιουργήθηκε η μεταβλητή *mutex*:

```
//our mutex  
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

η οποία, με χρήση των εντολών *pthread_mutex_lock(&mutex)* και *pthread_mutex_unlock(&mutex)*, κλειδώνει(*lock*) και ξεκλειδώνει(*unlock*) τις κρίσιμες περιοχές (Κ.Π.) των συναρτήσεων *db_add()* και *db_get()* αντίστοιχα, προκειμένου να επιτρέπεται η πρόσβαση μόνο σε ένα νήμα (όταν δημιουργηθούν) να εκτελέσει μία από τις δύο λειτουργίες κάθε φορά. Πιο συγκεκριμένα,

Στη συνάρτηση *db_add()*:

```
int db_add(DB* self, Variant* key, Variant* value)  
{  
    pthread_mutex_lock(&mutex); //kleidoma krisimis perioxis  
    if (memtable_needs_compaction(self->memtable))  
    {  
        INFO("Starting compaction of the memtable after %d insertions and %d deletions",  
             self->memtable->add_count, self->memtable->del_count);  
        sst_merge(self->sst, self->memtable);  
        memtable_reset(self->memtable);  
    }  
  
    //topothetisi tis memtable add stin krisimi perioxhi  
    //skopos: min ginetai taytoxrona prosthiki neon stoxeion kai anagnosi apo tin db_get  
    int res = memtable_add(self->memtable, key, value);  
    pthread_mutex_unlock(&mutex); //xekleidoma krisimis perioxis  
  
    return res; //epistrefetai i timi tis memtable_add meso tis res meta tin exodo apo tin kp  
}
```

Όπως φαίνεται και παραπάνω, με τις προαναφερθείσες εντολές κλειδώνεται και ξεκλειδώνεται η Κ.Π. της συνάρτησης. Επίσης, προκειμένου να μην γίνεται εισαγωγή νέων εγγγραφών στο *memtable* ενώ παράλληλα πραγματοποιείται ανάγνωση των ήδη υπάρχοντων με την *db_get()*, η συνάρτηση *memtable_add()* τοποθετείται μέσα στην Κ.Π. και η επιστρεφόμενη τιμή της ανατίθεται στη μεταβλητή *res* τύπου *int*, όπου, μέσω αυτής, επιστρέφεται μετά την έξοδο από την κρίσιμη περιοχή. Στη συνάρτηση *db_get()*:

```
int db_get(DB* self, Variant* key, Variant* value)
{
    pthread_mutex_lock(&mutex); //kleidoma krisimis perioxis
    if (memtable_get(self->memtable->list, key, value) == 1)
        //topothetisi enos extra unlock se periptosi poy to if ginei true na meinei locked to mutex
        pthread_mutex_unlock(&mutex);
        return 1;

    //topothetisi tis sst_get stin krisimi perioxi
    //skopos: min ginetai taytoxrona anagnosi stoixeion sto sst kai merging toy memtable me to disko
    int res = sst_get(self->sst, key, value);
    pthread_mutex_unlock(&mutex); //xekleidoma krisimis perioxis

    return res; //epistrefetai i timi tis sst_get meso tis res meta tin exodo apo tin kp
}
```

Παρόμοια τακτική ακολουθήθηκε κι εδώ με κλείδωμα/ξεκλείδωμα Κ.Π., όπου, τώρα εδώ, εισήχθηκε η συνάρτηση *sst_get()* προκειμένου να μην γίνεται ανάγνωση στοιχείων από το αρχείο *sst* στο δίσκο ενώ παράλληλα έχει ξεκινήσει συγχώνευση (*merging*) των εγγγραφών του *memtable* με το δίσκο. Επιπρόσθετα, τοποθετείται η εντολή *pthread_mutex_unlock(&mutex)* στο *block* κώδικα του *if* ώστε να μην παραμείνει κλειδωμένη η κρίσιμη περιοχή σε περίπτωση που η συνθήκη του αποτιμηθεί σε *true*.

2.2 Επεξεργασία αρχείου *db.c* - επίλυση προβλήματος σύγκρουσης μεταξύ νημάτων της μηχανής

Ένα ζήτημα που προκύπτει και πρέπει να αντιμετωπιστεί είναι το ενδεχόμενο σύγκρουσης μεταξύ των ήδη υπάρχοντων νημάτων στη μηχανή, αυτού που καλεί μια διαδοχική ακολουθία λειτουργιών *add* και *get* (νήμα εφαρμογής) και αυτού που είναι υπεύθυνο για τη σύμπτυξη *compaction* αρχείων μεταξύ των επιπέδων του *sst* όποτε κρίνεται απαραίτητο, στη μηχανή (νήμα μηχανής). Για το λόγο αυτό, στη συνάρτηση *db_add()*, προστίθεται η εξής εντολή:

```
while(pthread_mutex_trylock(&self->sst->cv_lock) == EBUSY){ //oso to mutex cv_lock tis merge_thread, min kaneis kati
    ;
}
pthread_mutex_unlock(&self->sst->cv_lock); //xekleidoma mutex cv_lock otan den ginetai merging
}
```

που σηματοδοτεί ότι όσο το *mutex cv_lock* είναι κλειδωμένο από τη *merge_thread()*, η οποία εκτελεί την διαδικασία της συγχώνευσης, η συνάρτηση *pthread_mutex_trylock()* επιστρέφει μια ειδική τιμή *EBUSY* στο χρήστη, ενημερώνοντάς τον ότι δεν έχει τελειώσει η διαδικασία της συγχώνευσης και πρέπει να περιμένει μέχρι το τέλος της. Έτσι, παραμένει στο *while()* όσο εκτελείται η *sst.merge()*, χωρίς να εκτε-

λείται κάτι μέσα στο βρόχο, ενώ όταν τελειώνει, γίνεται *unlock* με την εντολή *pthread_mutex_unlock(&self->sst->cv_lock)*, καθώς η *trylock* όταν το *cv_lock* είναι ξεκλειδωτό, το κλειδώνει κάτι που δεν πρέπει να συμβαίνει όσο δεν εκτελείται συγχώνευση.

```
int db_add(DB* self, Variant* key, Variant* value)
{
    pthread_mutex_lock(&mutex); //kleidoma krisimis perioxis
    if (memtable_needs_compaction(self->memtable))
    {
        INFO("Starting compaction of the memtable after %d insertions and %d deletions",
            self->memtable->add_count, self->memtable->del_count);
        sst_merge(self->sst, self->memtable);
        memtable_reset(self->memtable);
    }
    while(pthread_mutex_trylock(&self->sst->cv_lock) == EBUSY){ //oso to mutex cv_lock tis merge_thread, min kaneis kati
        ;
    }
    pthread_mutex_unlock(&self->sst->cv_lock); //xekleidoma mutex cv_lock otan den ginetai merging

    //topothetisi tis memtable add stin krisimi perioxi
    //skopos: min ginetai taytoxrona prosthiki neon stoiceion kai anagnosi apo tin db_get
    int res = memtable_add(self->memtable, key, value);
    pthread_mutex_unlock(&mutex); //xekleidoma krisimis perioxis

    return res; //epistrefetai i timi tis memtable_add meso tis res meta tin exodo apo tin kp
}
```

3 Επεξεργασία/Τροποποίηση κώδικα - Προετοιμασία για προσθήκη νημάτων - Λειτουργία *readwrite*

3.1 Επεξεργασία αρχείου *kiwi.c* - τροποποίηση *_write_test()*, *_read_test()*

Στο στάδιο αυτό, πραγματοποιούνται αλλαγές στο αρχείο *kiwi.c* με σκοπό την προσθήκη νημάτων που θα γίνει παρακάτω.

Ξεκινώντας, δημιουργήθηκαν αντίγραφα των αρχικών συναρτήσεων, *my_write_test()*, *my_read_test()* προκειμένου να τροποποιηθούν οι "υπογραφές" τους (επιστρεφόμενος τύπος, σύνολο ορισμάτων) ώστε να είναι συμβατές με το πρότυπο της συνάρτησης *pthread_create()* για τη δημιουργία νημάτων που θα χρησιμοποιηθεί παρακάτω. Συγκεκριμένα, έγιναν οι εξής αλλαγές:

Για *write* :

```
void * my_write_test(void *arg)
```

Για *read* :

```
void * my_read_test(void *arg)
```


Ενώ, τα ορίσματα των συναρτήσεων διατηρούνται σε δύο *structs*, ένα για την κάθε συνάρτηση:

```
//our structs for write&read functions|
struct kiwi_write{

    long int count;
    int r;
    DB* db;

};

struct kiwi_read{

    long int count;
    int r;
    DB* db;

};
```

Όπως φαίνεται και παραπάνω, στα *structs*, προστέθηκε και ο δείκτης **db* που είναι υπεύθυνος για το άνοιγμα και κλείσιμο της βάσης, του οποίου η χρήση θα εξηγηθεί αργότερα.

Ύστερα, δημιουργήθηκαν δείκτες σε κάθε *struct*, κάνοντας αρχικά το απαραίτητο *type casting*:

```
//we changed write test so it gets accepted in pthread create
void * my_write_test(void *arg)
{
    struct kiwi_str *wr = (struct kiwi_str*)arg; //casting sto orisma tis mywrite
}

//we changed read test so it gets accepted in pthread create
void * my_read_test(void *arg)
{
    struct kiwi_str *re = (struct kiwi_str*)arg;
}
```

με σκοπό τη δυνατότητα πρόσβασης και τροποποίησης των αρχικών ορισμάτων στο *block* κώδικα της κάθε συνάρτησης ως εξής:

my_write_test() :

```
for (i = 0; i < wr->count; i++) {
    if (wr->r)
        _random_key(key, KSIZE);
    else
        snprintf(key, KSIZE, "key-%d", i);
    fprintf(stderr, "%d adding %s\n", i, key);
    snprintf(val, VSIZE, "val-%d", i);

    sk.length = KSIZE;
    sk.mem = key;
    sv.length = VSIZE;
    sv.mem = val;

    db_add(wr->db, &sk, &sv);
    if ((i % 10000) == 0) {
        fprintf(stderr, "random write finished %d ops%30s\r",
                i,
                "");
        fflush(stderr);
    }
}
```

my_read_test() :

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

    /* if you want to test random write, use the following */
    if (re->r)
        _random_key(key, KSIZE);
    else
        snprintf(key, KSIZE, "key-%d", i);
    fprintf(stderr, "%d searching %s\n", i, key);
    sk.length = KSIZE;
    sk.mem = key;
    ret = db_get(re->db, &sk, &sv);
    if (ret) {
        //db_free_data(sv.mem);
        found++;
    } else {
        INFO("not found key#%s",
            sk.mem);
    }
}
```

Τέλος, αφαιρέθηκαν τα στατιστικά απόδοσης από τις συναρτήσεις των *read*, *write* και προστέθηκαν στην συνάρτηση *print_statistics()* που θα χρησιμοποιηθεί αργότερα.

```

//our functions, for finding statistics for read&write
void print_statistics(char * mode, double cost, void* arg1,int wp, int rp){

    if(strcmp(mode,"write")==0){
        struct kiwi_write *wr = (struct kiwi_write*)arg1;
        printf(LINE);
        printf("Write Statistics\n");
        printf("Total Number of Writes:%ld\n", wr->count);
        printf("%.6f sec/op; %.1f writes/sec(estimated); cost:%.3f(sec);\n"
            ,cost /(double) wr->count
            ,wr->count /(double) cost
            ,cost);

    }else if(strcmp(mode,"read")==0){
        struct kiwi_read *re = (struct kiwi_read*)arg1;
        printf(LINE);
        printf("Read Statistics\n");
        printf("Total Number of Reads:%ld\n", re->count);
        printf("%.6f sec/op; %.1f reads/sec(estimated); cost:%.3f(sec);\n"
            ,cost /(double) re->count
            ,re->count /(double) cost
            ,cost);

    }else if(strcmp(mode,"readwrite")==0){
        struct kiwi_write *rw = (struct kiwi_write*)arg1;
        double rper = (rp*(rw->count))/100;
        double wper = (wp*(rw->count))/100;
        printf("Read Statistics\n");
        printf("Total Number of Reads:%f\n", rper);
        printf("%d%% of the total number of reads read:%f\n",rp, rper);
        printf("%.6f sec/op; %.1f reads/sec(estimated); cost:%.3f(sec);\n"
            ,cost /(double) rper
            ,rper /(double) cost
            ,cost);
        printf("Write Statistics\n");
        printf("Total Number of Writes:%ld\n", rw->count);
        printf("%d%% of the total number of reads read:%f\n",wp, wper);
        printf("%.6f sec/op; %.1f reads/sec(estimated); cost:%.3f(sec);\n"
            ,cost /(double) wper
            ,wper /(double) cost
            ,cost);
    }
}

```

όπως και οι μεταβλητές *start*, *end*, *cost* που αφορούν την καταμέτρηση των στατιστικών απόδοσης, αντίστοιχα.

Συνολικά, η τελική μορφή των συναρτήσεων *my_write_test()*, *my_read_test()* φαίνεται παρακάτω:

my_write_test() :

```
void * my_write_test(void *arg)
{
    struct kiwi_write *wr = (struct kiwi_write*)arg;
    int i;
    Variant sk, sv;

    char key[KSIZE + 1];
    char val[VSIZE + 1];
    char sbuf[1024];

    memset(key, 0, KSIZE + 1);
    memset(val, 0, VSIZE + 1);
    memset(sbuf, 0, 1024);

    for (i = 0; i < wr->count; i++) {
        if (wr->r)
            _random_key(key, KSIZE);
        else
            snprintf(key, KSIZE, "key-%d", i);
        fprintf(stderr, "%d adding %s\n", i, key);
        snprintf(val, VSIZE, "val-%d", i);

        sk.length = KSIZE;
        sk.mem = key;
        sv.length = VSIZE;
        sv.mem = val;

        db_add(wr->db, &sk, &sv);
        if ((i % 10000) == 0) {
            fprintf(stderr, "random write finished %d ops%30s\r",
                    i,
                    "");
            fflush(stderr);
        }
    }

    return NULL;
}
```

my_read_test() :

```
void * my_read_test(void *arg)
{
    struct kiwi_read *re = (struct kiwi_read*)arg;
    int i;
    int ret;
    int found = 0;
    Variant sk;
    Variant sv;
    char key[KSIZE + 1];

    for (i = 0; i < re->count; i++) {
        memset(key, 0, KSIZE + 1);

        /* if you want to test random write, use the following */
        if (re->r)
            _random_key(key, KSIZE);
        else
            snprintf(key, KSIZE, "key-%d", i);
        fprintf(stderr, "%d searching %s\n", i, key);
        sk.length = KSIZE;
        sk.mem = key;
        ret = db_get(re->db, &sk, &sv);
        if (ret) {
            //db_free_data(sv.mem);
            found++;
        } else {
            INFO("not found key#%s",
                sk.mem);
        }

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

    return NULL;
}
```

3.2 Επεξεργασία αρχείου *bench.c* - Προσαρμογή αλλαγών από το αρχείο *kiwi.c*

Αρχικά, προστέθηκαν τα πρότυπα των τροποποιημένων/νέων συναρτήσεων του *kiwi.c* πάνω από τη *main*, ώστε να είναι ορατά σε αυτή:

```
//function prototype
void * my_write_test(void *arg);
void * my_read_test(void *arg);
void print_statistics(char * mode, double cost, void* arg1);
```

όπως και τα *structs* των ορισμάτων των συναρτήσεων *my_write_test()*, *my_read_test()*, για τον ίδιο λόγο:

```
//our structs for write&read functions
struct kiwi_write{

    long int count;
    int r;
    DB* db;

};

struct kiwi_read{

    long int count;
    int r;
    DB* db;

};
```

3.3 Επεξεργασία αρχείου *bench.c* - Βελτιστοποίηση λειτουργιών *write*, *read*, προσθήκη λειτουργίας *readwrite*

Στη συνέχεια, σε κάθε μία από τις υπάρχουσες λειτουργίες *read*, *write*, μεταφέρεται η λειτουργία του ανοίγματος και κλείσιμου της βάσης από το αρχείο *kiwi.c*.

Πρώτα, στο αρχείο *bench.c*, συμπεριλαμβάνονται τα εξής:

```
#include "../engine/db.h" //aparaitita gia anoigma/klisimo vasis apo to bench.c
#define DATAS ("testdb") I
```

Αντίστοιχα, προστέθηκαν και οι μεταβλητές *start*, *end*, *cost* που αφορούν την καταμέτρηση των στατιστικών απόδοσης στη *main* του *bench.c*, οι οποίες θα χρησιμοποιηθούν για να μετρηθεί αργότερα το κόστος, μιας και το άνοιγμα/κλείσιμο της βάσης θα πραγματοποιείται από εκεί:

```
int main(int argc, char** argv)
{
    long long start,end; //using these variables to measure time
    double cost;
```

Στην περίπτωση που ο χρήστης πληκτρολογήσει "write":

Αρχικά, προστίθεται ένας δείκτης **wr*, με το απαραίτητο *type casting*, ως εξής:

```
3 int main(int argc, char** argv)
4 {
5     long long start, end; //using these variables to measure time
6     double cost;
7     //global variable because we need only one writer
8     struct kiwi write *wr = (struct kiwi write*)malloc(sizeof(struct kiwi write));
```

ο οποίος θα είναι *global*, καθώς στην ταυτόχρονη λειτουργία *read*, *write* που θα προστεθεί παρακάτω, πολυνηματικός κώδικας πρόκειται να εφαρμοστεί μόνο στους αναγνώστες.

Όπως φαίνεται και παρακάτω, με τις *wr* → *db* = *db_open(DATAS)*; , *db_close(wr* → *db)*; ανοίγει και κλείνει η βάση αντίστοιχα ενώ, με τη χρήση της εντολής *start* = *get_ustime_sec()*; στην αρχή του *block* κώδικα του *if*, της *end* = *get_ustime_sec()*; στο τέλος και ύστερα με την *cost* = *end* − *start*;, υπολογίζεται ο χρόνος που απαιτείται προκειμένου να ολοκληρωθεί η διαδικασία της εγγραφής των στοιχείων στη δομή(κόστος), όσο είναι ανοιχτή η βάση. Επίσης, με χρήση του δείκτη *wr* του *struct* για το *write*, αρχικοποιούνται κατάλληλα τα πεδία του ώστε να χρησιμοποιηθούν όταν δίνονται ως παραμέτροι κατά την κλήση της *_my_write_test* μέσω του *struct* ως όρισμα. Τέλος, με την *print_statistics("write", cost, wr)*;, τυπώνονται τα στατιστικά των εγγραφών, δηλαδή πόσες εγγραφές πραγματοποιήθηκαν ανά δευτερόλεπτο, η ρυθμαπόδοση και το κόστος.

```
if (strcmp(argv[1], "write") == 0) {
    start = get_ustime_sec(); //get start measuring system time
    wr->count = atoi(argv[2]); //string to int
    wr->db = db_open(DATAS); //opening data base
    _print_header(wr->count);
    _print_environment();
    if (argc == 4){
        wr->r = 1;
    }
    my_write_test(wr); //calling fuction my_write_test
    db_close(wr->db); //closing data base
    end = get_ustime_sec(); //get end measuring system time
    cost = end - start; //calculating the cost
    print_statistics("write", cost, wr, 0, 0); //printing statistics
    free(wr);
}
```

Αντίστοιχη τακτική ακολουθήθηκε και στην περίπτωση που ο χρήστης πληκτρολογήσει "read". Εδώ, ωστόσο, ο δείκτης **re* είναι *local*, λόγω της πολυνηματικής υλοποίησης των αναγνωστών στην ταυτόχρονη λειτουργία *read*, *write*.

```

        free(wr);
    } else if (strcmp(argv[1], "read") == 0) {
        struct kiwi_read *re = (struct kiwi_read*)malloc(sizeof(struct kiwi_read));
        start = get_ustime_sec();
        re->count = atoi(argv[2]);
        re->db = db_open(DATAS);
        _print_header(re->count);
        _print_environment();
        if (argc == 4)
            re->r = 1;
        my_read_test(re);
        db_close(re->db);
        end = get_ustime_sec();
        cost = end - start;
        print_statistics("read", cost, re, 0, 0);
        free(re);
    }
}

```

Επίσης, επιχειρήθηκε και η προσθήκη μίας ταυτόχρονης λειτουργίας *read, write*, χωρίς όμως τα επιθυμητά αποτελέσματα*. Παρόλ'αυτά, υλοποιήθηκαν τα εξής: Σε πρώτη φάση, συμπεριλήφθηκε στο αρχείο *db.h*, η βιβλιοθήκη *pthread.h* ως εξής:

```
#include <pthread.h>
```

Όπως φαίνεται και παρακάτω, αρχικά, όπως και στις μεμονωμένες λειτουργίες *read, write*, χρησιμοποιούνται οι μεταβλητές *start, end, cost* και ο δείκτης **db* για την καταμέτρηση των στατιστικών απόδοσης και το άνοιγμα/κλείσιμο της βάσης αντίστοιχα. Στη συνέχεια, με τις μεταβλητές *writeper, readper* και τη χρήση της *atoi()*, λαμβάνονται τα ποσοστά που θα εκτελέσει η εκάστοτε λειτουργία από τον αρχικό αριθμό των εγγραφών και με κατάλληλες πράξεις στις μεταβλητές *count* των *structs read* και *write*, ανατίθεται στην κάθε λειτουργία ο ακριβής αριθμός εγγραφών που θα προσθέσει στη ή θα διαβάσει από τη δομή αντίστοιχα. Στη συνέχεια, για τη λειτουργία *write*, δημιουργείται η μεταβλητή *write*, τύπου *pthread_t*, λόγω του ενός γραφέα και ο πίνακας *read[2]*, τύπου *pthread_t*, (για 2 νήματα αρχικά σε πειραματικό στάδιο), λόγω των πολλαπλών αναγνωστών. Έπειτα, με τις συναρτήσεις *pthread_create(&write, NULL, my_read_test, (void*)wr)* και *pthread_create(&read[i], NULL, my_read_test, (void*)re);, i = 1...2*, *pthread_join(write, NULL);* και *pthread_join(read[i], NULL);, i = 1...2* δημιουργούνται και αποδεσμεύουν τους πόρους κατά τον τερματισμό τους αντίστοιχα, τα νήματα. Στο μεταξύ, διατηρείται σε ένα *if*, το μήνυμα όπου το σύστημα θα ενημερώσει το χρήστη, σε περίπτωση λανθασμένου αριθμού ορισμάτων ή εντολής στο τερματικό. Τέλος, με την *print_statistics* ("readwrite", *cost, re, writeper, readper*);, τυπώνονται τα στατιστικά των εγγραφών, δηλαδή πόσες εγγραφές πραγματοποιήθηκαν ανά δευτερόλεπτο, την ρυθμική απόδοση και το κόστος ενώ έχει τροποποιηθεί κατάλληλα και η συνθήκη *else* για το λανθασμένο τρόπο ορισμάτων και για την ταυτόχρονη λειτουργία *read&write*.

```

    } else if(strcmp(argv[1], "readwrite") == 0)
    {

        if(argc<4){
            fprintf(stderr,"Usage: db-bench <readwrite> <count> <percentageofwrites> <percentageofreads>\n");
            exit(1);
        }
        start = get_ustime_sec();
        struct kiwi_read *re = (struct kiwi_read*)malloc(sizeof(struct kiwi_read));
        wr->r = 0;
        re->r = 0;
        int writeper = atoi(argv[3]);
        int readper = atoi(argv[4]);

        DB* db = db_open(DATAS);
        wr->db = db;
        re->db = db;
        long long count = atol(argv[2]);
        wr->count = (writeper*count)/100;
        re->count = (readper*count)/100;
        _print_header(count);
        _print_environment();

        pthread_t write;
        pthread_t read[2];

        pthread_create(&write, NULL, my_write_test, (void*) wr);
        for(int i=0;i<2;i++){
            pthread_create(&read[i], NULL, my_read_test, (void*) re);
        }
        pthread_join(write, NULL);
        for(int i=0;i<2;i++){
            pthread_join(read[i], NULL);
        }

        db_close(re->db);
        end = get_ustime_sec();
        cost = end - start;
        //print_statistics("readwrite", cost, re,writeper,readper);
        free(wr);
        free(re);
    }
    } else {
        fprintf(stderr,"Usage: db-bench <write | read> | <readwrite> <count> <random>\n");
        exit(1);
    }
}

```

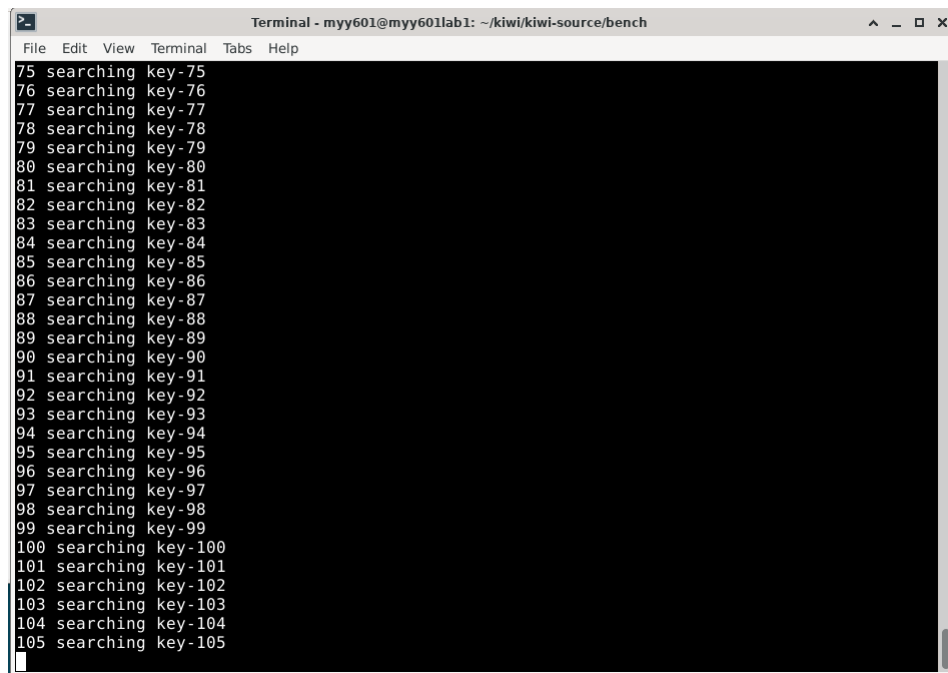
4 Έξοδος τελικής εντολής *make all*

```

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

```


*Το πρόγραμμα έτρεχε χωρίς *error* όμως αυτά, σε ένα σημείο ενώ εκτελούνταν, κολλούσε:

A screenshot of a terminal window titled "Terminal - myy601@myy601lab1: ~/kiwi/kiwi-source/bench". The terminal displays a list of 31 lines, each starting with a number followed by the word "searching" and a key identifier. The keys range from key-75 to key-105, with some numbers missing (e.g., 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105). The terminal has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The background is black, and the text is white. The window has standard Linux window controls (minimize, maximize, close) in the top right corner.

```
75 searching key-75
76 searching key-76
77 searching key-77
78 searching key-78
79 searching key-79
80 searching key-80
81 searching key-81
82 searching key-82
83 searching key-83
84 searching key-84
85 searching key-85
86 searching key-86
87 searching key-87
88 searching key-88
89 searching key-89
90 searching key-90
91 searching key-91
92 searching key-92
93 searching key-93
94 searching key-94
95 searching key-95
96 searching key-96
97 searching key-97
98 searching key-98
99 searching key-99
100 searching key-100
101 searching key-101
102 searching key-102
103 searching key-103
104 searching key-104
105 searching key-105
```

5 Πληροφορίες συστήματος

Πληροφορίες συστήματος στο οποίο υλοποιήθηκε η άσκηση:

```
CPU: 2 * AMD Ryzen 7 5700U with Radeon Graphics
```

Χρησιμοποιήθηκαν επίσης (για την δημιουργία του εξωφύλλου *Figma*, για τα γραφήματα *Canva* και για την δημιουργία της αναφοράς, *LaTeX*):



Canva

L^AT_EX