

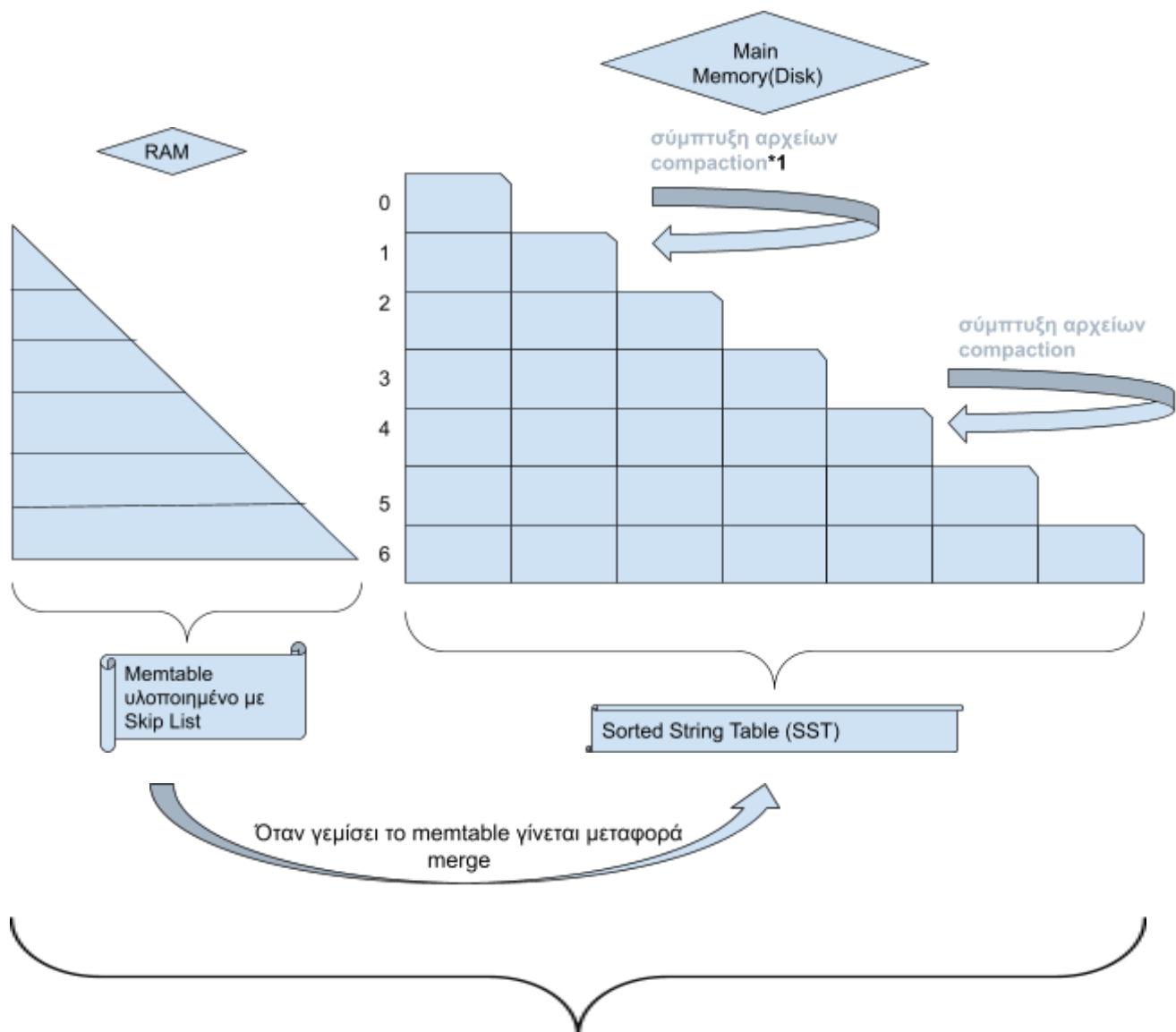
Λειτουργικά Συστήματα Lab1

Ιωάννης Τσόχλας
AM 4993

Κωνσταντίνος-Διονύσιος Λαμανιακού
AM 5110

Συνοπτική Επεξήγηση της δομής δεδομένων

Ο κώδικας που επεξεργαζόμαστε υλοποιεί την μηχανή αποθήκευσης Kiwi η οποία βασίζεται στην δομή δεδομένων LSM-tree, παρακάτω εξηγούμε συνοπτικά την δομή.



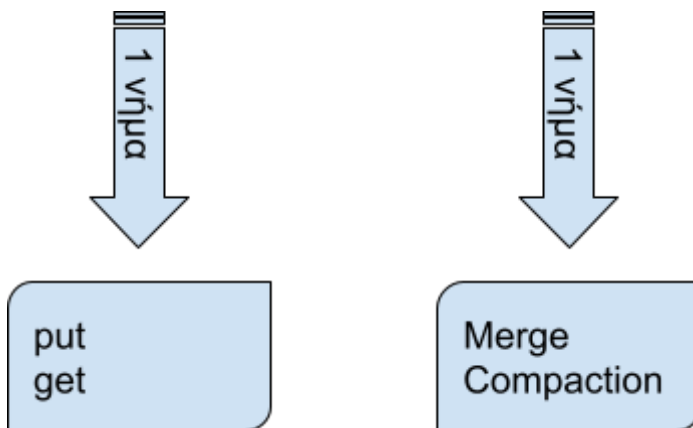
Δομή Δεδομένων

1

¹ Η σύμπτυξη αρχείων compaction ενεργοποιείται όταν το πλήθος των αρχείων σε ένα επίπεδο ξεπερνάει ένα προκαθορισμένο κατώφλι

Συνοπτική Επεξήγηση Νημάτων

Στην υλοποίηση που μας παρέχετε υπάρχουν ήδη δύο νήματα στη μηχανή αποθήκευσης



Η συνεισφορά μας στο project αποσκοπεί στο να δημιουργήσουμε μία πολυνηματική λειτουργία των εντολών put και get.

- Στην **1η Φάση** προσθέτουμε μία κλειδαριά για να εφαρμόσουμε αμοιβαίο αποκλεισμό στις λειτουργίες put και get .(Μόνο ένα νήμα θα γράφει ή θα διαβάζει)
- Στην **2η Φάση** επιτρέπουμε πολλαπλούς αναγνώστες ή ένα γραφέα να λειτουργεί κάθε φορά
- Στην **3η Φάση** επιτρέπουμε πολλαπλούς αναγνώστες και ένα γραφέα σε διαφορετικές δομές της βάσης (πχ, Memtable, sst)

1η Φάση

Χρησιμοποιούμε μία κλειδαριά για να εφαρμόσουμε αμοιβαίο αποκλεισμό στις λειτουργίες put και get.

Επεξήγηση :

Με την εφαρμογή της κλειδαριάς(pthread-mutex) Μόνο ένας ή θα γράφει ή θα διαβάζει (Θα έχει πρόσβαση στη βάση δεδομένων).

- ☒ Ο κώδικας που έχουμε τροποποίηση στο αρχείο db.c φαίνεται παρακάτω με **bold**

...

```
pthread_mutex_t mutex ; //create mutex for db_add db_get
```

```
int db_add(DB* self, Variant* key, Variant* value)
```

```
{
```

```
    pthread_mutex_lock(&mutex); //lock
```

```
    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);
```

```
    }
```

```
    pthread_mutex_unlock(&mutex); //unlock
```

```
    return memtable_add(self->memtable, key, value);
```

```
}
```

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

```
{
```

```
    pthread_mutex_lock(&mutex); //lock
```

```
    if (memtable_get(self->memtable->list, key, value) == 1)
```

```
        return 1;
```

```
    pthread_mutex_unlock(&mutex); //unlock
```

```
    return sst_get(self->sst, key, value);
```

```
}
```

...

Να αναφέρουμε ότι ο κώδικας που επεξηγούμε εδώ δεν είναι ο παραδοτέος διότι στην συνέχεια τον τροποποιούμε περαιτέρω

Write(add) before pthread_mutex

```
Random-Write (done:100000): 0.000020 sec/op; 50000.0 writes/sec(estimated); cost:2.000(sec);  
yy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Read(get) before pthread_mutex

```
|Random-Read (done:100000, found:100000): 0.000010 sec/op; 100000.0 reads /sec(estimated); cost:1.000(sec)  
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

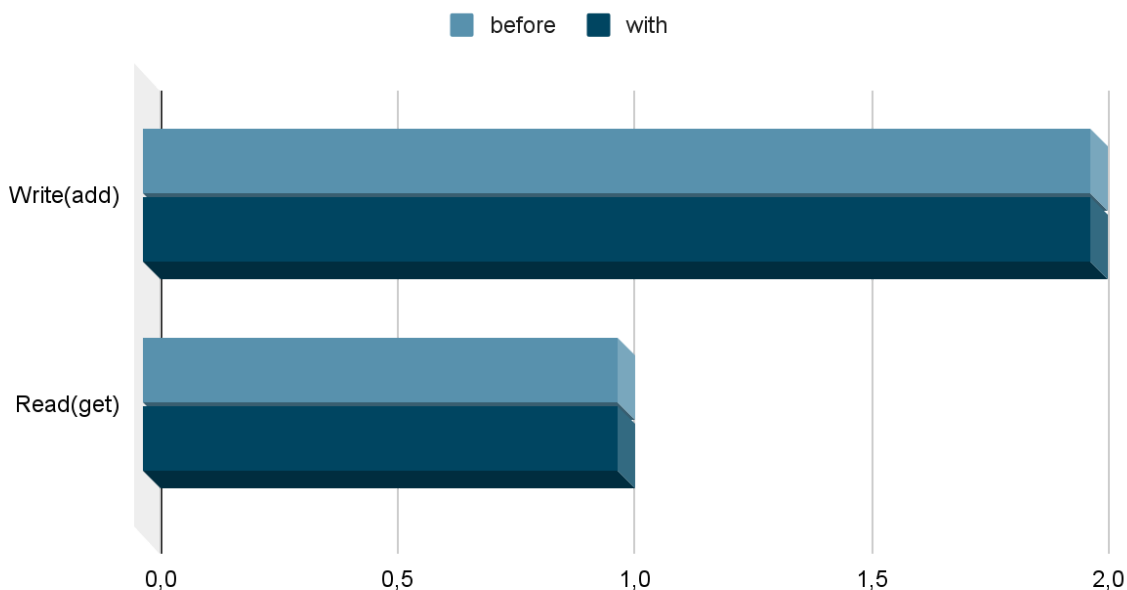
Write(add) with pthread_mutex

```
Random-Write (done:100000): 0.000020 sec/op; 50000.0 writes/sec(estimated); cost:2.000(sec);  
yy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Read(get) with pthread_mutex

```
|Random-Read (done:100000, found:100000): 0.000010 sec/op; 100000.0 reads /sec(estimated); cost:1.000(sec)  
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Μετάβολη Χρόνου

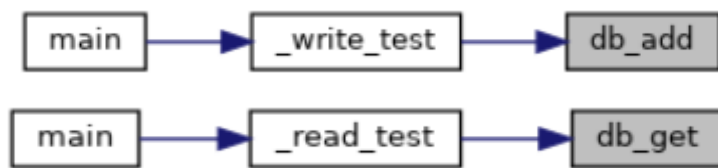


Επεξήγηση Μεταβολής χρόνου : Με τις κλειδαριές προστατεύουμε τον κώδικα που επιθυμούμε(Κρίσιμη περιοχή) από το να τον εκτελέσουν κι άλλα νήματα την ίδια χρονική στιγμή με αποτέλεσμα να αλλοιωθούν τα δεδομένα(προκαλέσουν ανεπιθύμητες αλλαγές), με την τροποποίηση βλέπουμε το χρόνο να είναι ίδιος , διότι το πρόγραμμά μας δεν είναι πολυνηματικό (Υπάρχει μόνο ένα νήμα το οποίο διατρέχει την add get , Συνεπώς δεν υπάρχει νήμα που να τίθεται σε αναμονή).

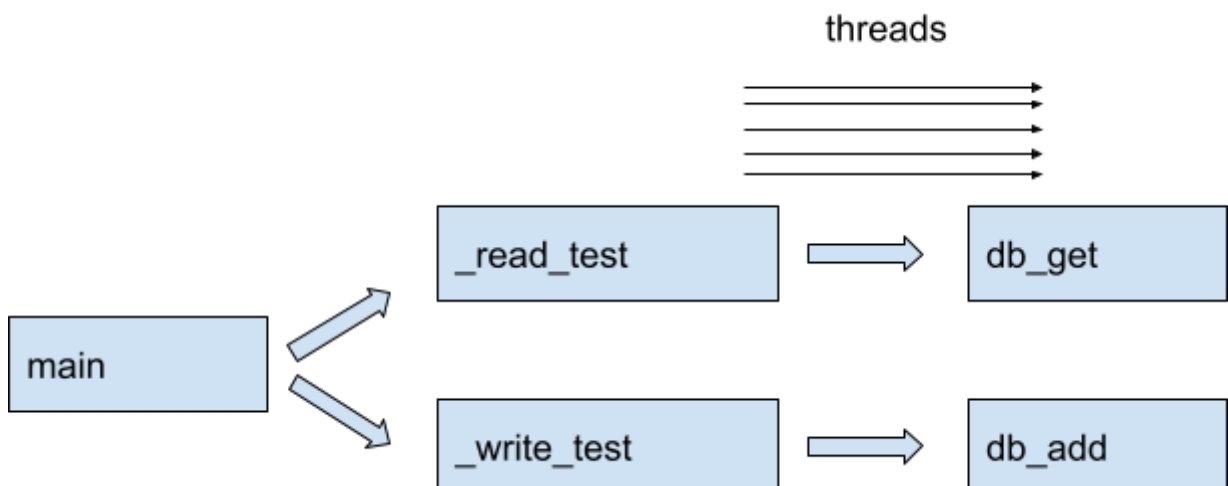
2η Φάση

Λειτουργούν πολλά νήματα για την εντολή `get(read)` ή ένα νήμα για την εντολή `add/put(write)` .

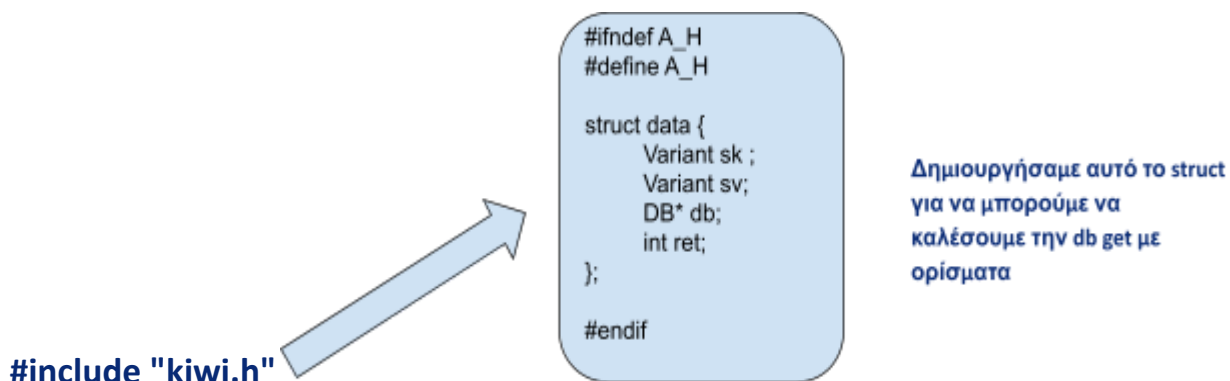
Παρατηρώντας τη τεκμηρίωση του κώδικα, συνειδητοποιήσαμε ότι τις εντολές `add`, `get` τις καλούν οι `_write_test`, `_read_test` αντίστοιχα.



Οπότε δημιουργήσαμε τα νήματα (`pthread_create`) στην συνάρτηση `_read_test`. Καθιστώντας με αυτό τον τρόπο την εντολή `get` πολυνηματική



- ☑ **Ο κώδικας που έχουμε προσθέσει στο αρχείο *kiwi.c* φαίνεται παρακάτω με *bold***



```
void _read_test(long int count, int r)
{
```

```
    pthread_t tid[5]; //create 5 threads
    struct data thread_args; //use struct in function
    int i;
    int found = 0;
    double cost;
    long long start,end;
    char key[KSIZE + 1];
    void *status ;//the variable for the return
```

```
    thread_args.db = db_open(DATAS); //arguments for db get forward to struct
    start = get_ustime_sec();
    for (i = 0; i < count; i++) {
        memset(key, 0, KSIZE + 1);
        /* if you want to test random write, use the following */
        if (r)
            _random_key(key, KSIZE);
        else
            snprintf(key, KSIZE, "key-%d", i);
        fprintf(stderr, "%d searching %s\n", i, key);
        thread_args.sk.length = KSIZE; //arguments for db get forward to struct
        thread_args.sk.mem = key; //arguments for db get forward to struct
```

```
int k;
for (k=0; k<5; k++){
    pthread_create(tid + k,NULL,db_get,(void *) &thread_args);//create thread
}
for(k=0; k<5; k++){
    pthread_join(tid[k],&status);//releasing thread resources
}
if ((intptr_t)status) //the return of db get
    //db_free_data(sv.mem);
    found++;
} else {
    INFO("not found key#%s",
        thread_args.sk.mem);
}

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

☒ Ο κώδικας που έχουμε τροποποιήσει στο αρχείο *db.c*

Χρησιμοποιήσαμε την δομή της διαφάνειας που επεξηγεί πώς να περάσουμε πολλές παραμέτρους στη συνάρτησή μας

```
void *db_get(void *arg)
{
    int x;
    struct data *d=(struct data *) arg;
    pthread_mutex_lock(&mutex); //lock
    if (memtable_get(d->db->memtable->list, &d->sk, &d->sv) == 1){
        return (void *)(intptr_t)1;
    }
    x=sst_get(d->db->sst, &d->sk, &d->sv);
    pthread_mutex_unlock(&mutex);//unlock
    return (void *)(intptr_t)x;
}
```


Πειράματα

Για αρχή θέλουμε να δούμε αν τα νήματα που δημιουργήσαμε λειτουργούν σωστά αυτο το καταφέραμε με αντίστοιχα μηνύματα οταν ξεκινάνε και οταν τερματίζουν :

```
for (k=0; k<5; k++){
    pthread_create(&tid[k],NULL,db_get,(void *) &thread_args);//createthread
    printf("Thread %d has started\n",k);
}
for(k=0; k<5; k++){
    pthread_join(tid[k],&status);
    printf("Thread %d has finished\n",k);
}
```

παρακάτω φαίνεται πως παρουσιάζεται στο τερματικό :

```
Thread 0 has started
Thread 1 has started
Thread 2 has started
Thread 3 has started
Thread 4 has started
Thread 0 has finished
Thread 1 has finished
Thread 2 has finished
Thread 3 has finished
Thread 4 has finished
```

Τα επόμενα πειράματα αφορούν τις αλλαγές στον χρόνο :

Κάναμε διάφορα πειράματα για να δούμε τι αλλαγές βλέπουμε στον χρόνο ανάλογα τα πόσα νήματα χρησιμοποιούμε :

1 thread:

```
|Random-Read (done:100000, found:100000): 0.000030 sec/op; 33333.3 reads /sec(estimated); cost:3.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

2 threads:

```
|Random-Read (done:100000, found:100000): 0.000040 sec/op; 25000.0 reads /sec(estimated); cost:4.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

3 threads:

```
|Random-Read (done:100000, found:100000): 0.000050 sec/op; 20000.0 reads /sec(estimated); cost:5.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

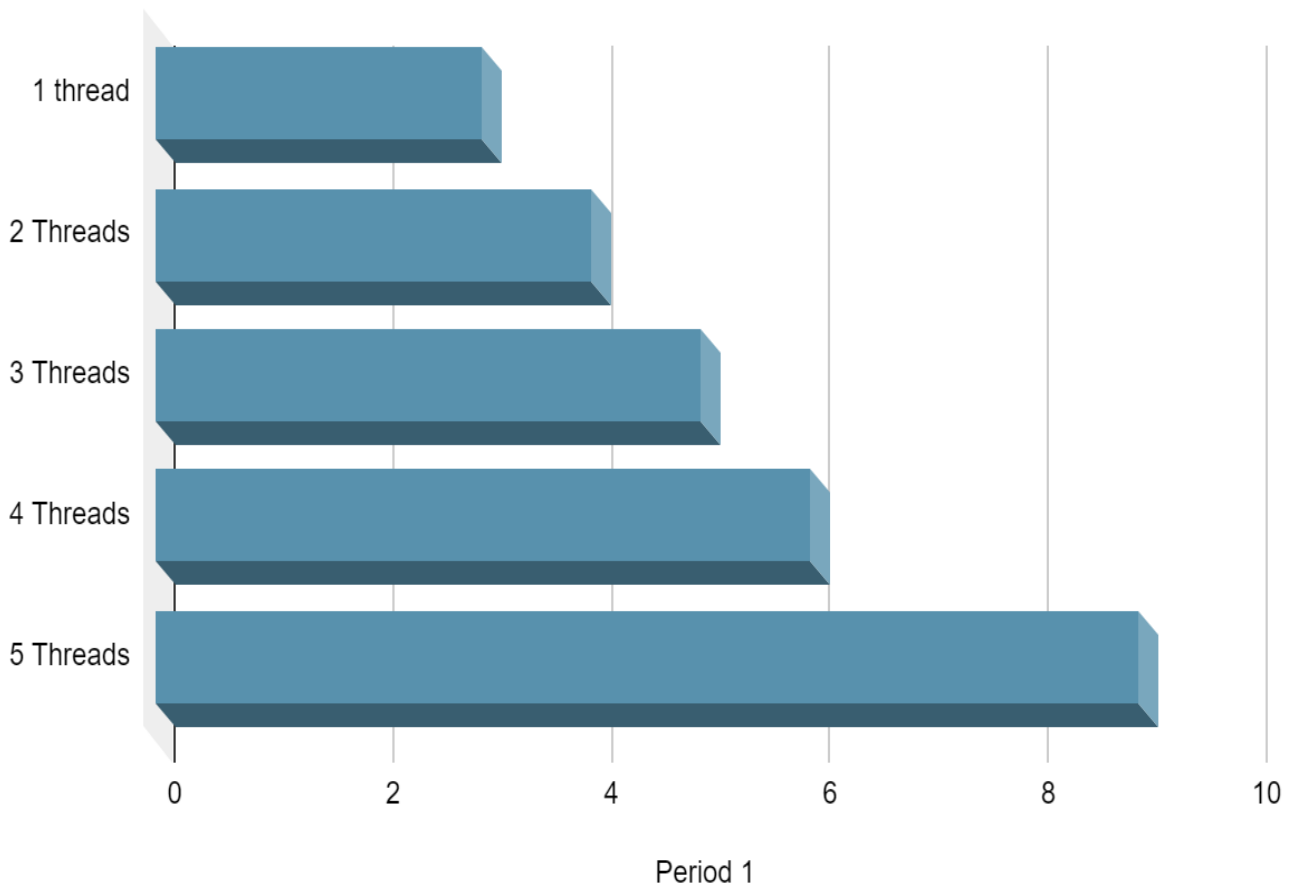
4 threads:

```
|Random-Read (done:100000, found:100000): 0.000060 sec/op; 16666.7 reads /sec(estimated); cost:6.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

5 threads:

```
|Random-Read (done:100000, found:100000): 0.000090 sec/op; 11111.1 reads /sec(estimated); cost:9.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Διαγραμμα Μεταβολής χρόνου :



Παρατηρησεις : Βλέπουμε μία αύξηση του χρόνου όσο αυξάνουμε τον αριθμό των Threads αυτο συμβαίνει διότι τρέχουμε παράλληλα περισσότερες φορές τον ίδιο κώδικα .Με αυτον τρόπο καταφέρνουμε περισσότεροι χρήστες να μπορούν να διαβάσουν ταυτόχρονα.

Τελικο test για την 2η φαση

Για να ολοκληρώσουμε την 2η φαση, δηλαδή να λειτουργούν πολλά νήματα για την εντολή `get(read)` ή ένα νήμα για την εντολή `add/put(write)`

δημιουργήσαμε την `readwrite_v2` όπου θα επιλέγει τυχαία ποιά απο τις δυο λειτουργίες θα εκτελεί κάθε φορά.

- ☒ Ο κώδικας που έχουμε προσθέσει στο αρχείο `kiwi.c` φαίνεται παρακάτω με ***bold***

```
void _readwrite_v2(int count,int r,int nimata){
    srand(time(NULL)); //makes use of the computer's internal clock to control the choice of the seed
    int n = rand() % 2; //if odd make it 0 else 1
    if (n==0){
        _write_test(count,r,1);
    }else{
        _read_test(count,r,nimata);
    }
}
```

- ☒ Ο κώδικας που έχουμε προσθέσει στο αρχείο `bench.c` φαίνεται παρακάτω με ***bold***

```
else if(strcmp(argv[1], "readwrite_v2") == 0){
    int r = 0;
    int nimata;//we use it to take the number of threads from main arg
    count = atoi(argv[2]);
    _print_header(count);
    _print_environment();
    if (argc == 5) //we change this from 4 to 5
        r = 1;
    if (argc == 4){ //the argument 4 exist we have threads from console
        nimata=atoi(argv[3]); //take the number of threads and make it int with atoi
    }else{
        nimata=1; //else we dont have threads from console and take the default to 1
    }
    _readwrite_v2(count, r,nimata); //put the number inside of write test to use it on
db_get
}
```

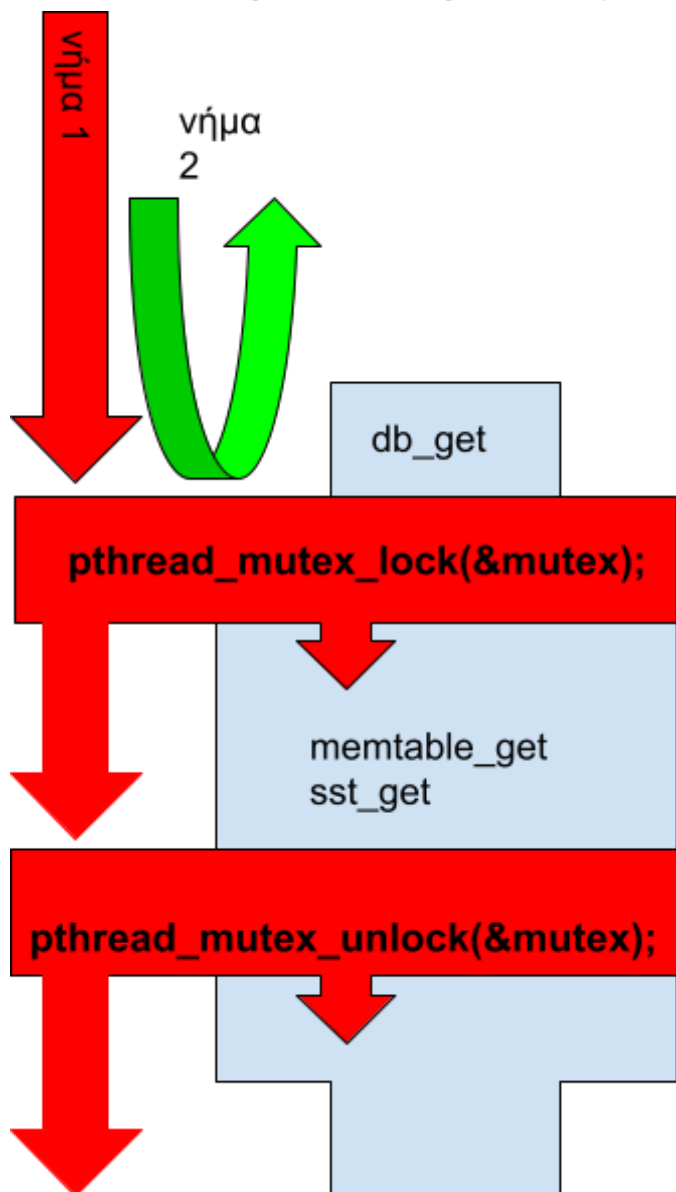
Για να τρέξουμε το Test στο τερματικο αρκει να γραψουμε την παρακατω εντολη στο τερματικο(οπου 2 ο αριθμος τον νηματων) :

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite_v2 100000 2
```

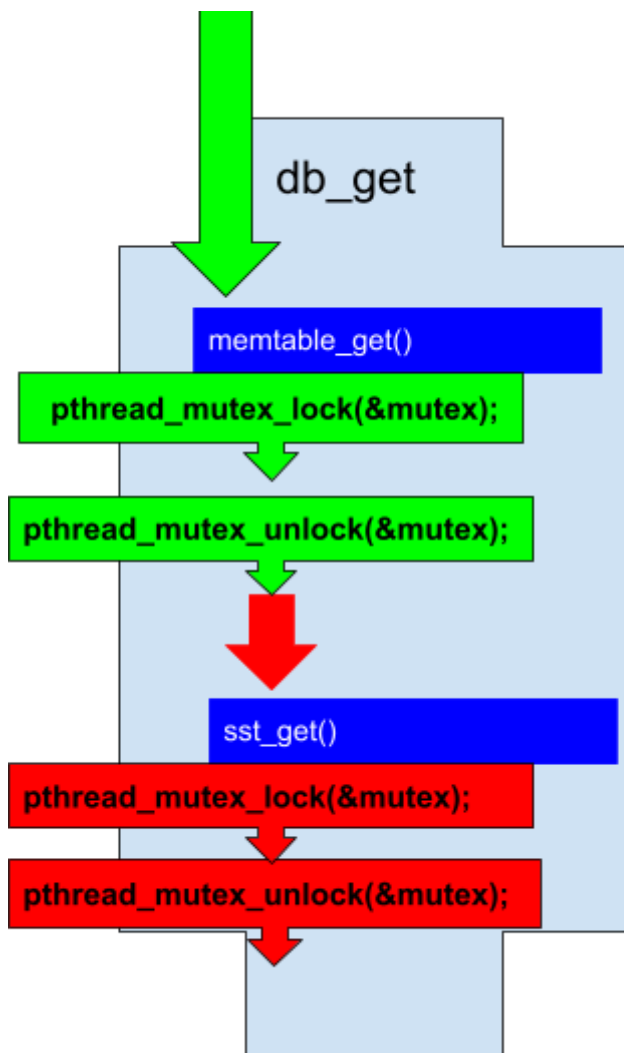
3η Φάση

Βελτίωση χρόνου χρησιμοποιώντας διαφορετικές κλειδαριές για το memtable και διαφορετικές για το sst

Η κλειδαριά που βάλαμε στην πρώτη φάση επιτρέπει μόνο ένα νήμα τη φορά να μπαίνει μέσα στον κρίσιμο κώδικα της `db_get`. Το νήμα 1 προλαβαίνει και μπαίνει στο κρίσιμο κώδικα της `db_get` απαγορεύοντας με αυτό τον τρόπο την πρόσβαση σε άλλα νήματα να μπουν ταυτόχρονα μέσα σε αυτό. Όπως είναι λογικό υπάρχει χρονοτριβή καθώς τα νήματα περιμένουν τις κλειδαριές να ανοίξουν .



Αρχικά αφαιρέσαμε τις κλειδαριές από την `db_get`. Με το σκεπτικό να προσθέσουμε κλειδαριές στις καλούσες συναρτήσεις(`memtable_get`,`sst_get`),ώστε διαφορετικά νήματα να λειτουργούν ταυτόχρονα εφόσον δεν τροποποιούν το ίδιο μέρος του συστήματος το ίδιο χρονικό διάστημα.,Συγκεκριμένα από εκεί που ένα νήμα θα διατρέχει ακολουθιακά τις δύο συναρτήσεις τώρα όταν το πρώτο νήμα βγαίνει από την πρώτη θα εισέρχεται στη δεύτερη και θα αφήνει χώρο στην πρώτη να μπει ένα άλλο.



Η τροποποίηση που κάναμε είναι συγκεκριμένα στην συνάρτηση `memtable_get()` τοποθετώντας τις κλειδαριές που φαίνονται παρακάτω με **Bold**.

```
pthread_mutex_t mem;
int memtable_get(SkipList* list, const Variant *key, Variant* value)
{
    SkipNode* node = skiplist_lookup(list, key->mem, key->length);

    if (!node){
        return 0;
    }
    pthread_mutex_lock(&mem);
    const char* encoded = node->data;
    encoded += varint_length(key->length) + key->length;

    uint32_t encoded_len = 0;
    encoded = get_varint32(encoded, encoded + 5, &encoded_len);
    pthread_mutex_unlock(&mem);
    if (encoded_len > 1){
        buffer_putnstr(value, encoded, encoded_len - 1);
    }else{
        return 0;
    }
    return 1;
}
```

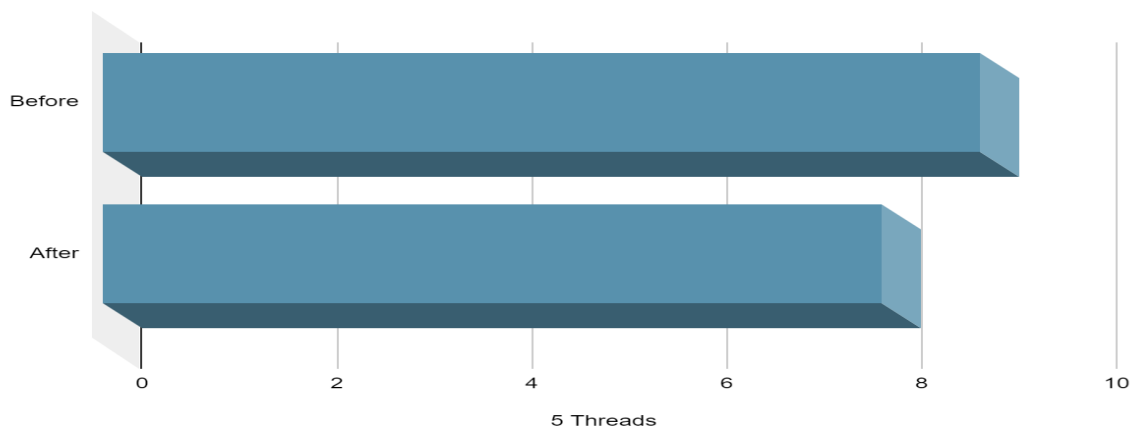
Η βελτίωση που βλέπουμε φαίνεται στο παρακάτω screenshot

5 Threads Read before:

```
|Random-Read      (done:100000, found:100000): 0.000090 sec/op; 11111.1 reads /sec(estimated); cost:9.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

5 Threads Read after:

```
|Random-Read      (done:100000, found:100000): 0.000080 sec/op; 12500.0 reads /sec(estimated); cost:8.000(sec)
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```



Threaded Benchmark

Δημιουργία πολλαπλών νημάτων με χρήση παραμέτρου από τη γραμμή εντολών

Αυτό το καταφέραμε προσθέτοντας στο αρχείο bench.c τις εντολές που φαίνονται παρακάτω με **Bold**.

```
if (strcmp(argv[1], "write") == 0) {  
    int r = 0;  
    int nimata; //we use it to take the number of threads from main arg  
    count = atoi(argv[2]);  
    _print_header(count);  
    _print_environment();  
    if (argc == 5) //we change this from 4 to 5  
        r = 1;  
    if (argc == 4) //the argument 4 exist we have threads from console  
        nimata=atoi(argv[3]); //take the number of threads and make it int with atoi  
    }else{  
        nimata=1; //else we don't have threads from console and take the default to 1  
    }  
    _write_test(count, r,nimata); //put the number inside of write test to use it on db_add  
} else if (strcmp(argv[1], "read") == 0) {  
    int r = 0;  
    int nimata; //we use it to take the number of threads from main arg  
    count = atoi(argv[2]);
```

```
_print_header(count);  
_print_environment();  
if (argc == 5) //we change this from 4 to 5  
    r = 1;  
if (argc == 4){ //the argument 4 exist we have threads from console  
    nimata=atoi(argv[3]); //take the number of threads and make it int with atoi  
}else{  
    nimata=1; //else we don't have threads from console and take the default to 1  
}  
_read_test(count, r,nimata); //put the number inside of write test to use it on db_get
```

και τροποποιώντας τον κωδικά στο kiwi.c

```
void _write_test(long int count, int r,int nimata)  
{  
    pthread_t tid[nimata]; //create  
    struct data thread_args; //struct  
    int i;  
    double cost;  
    long long start,end;  
    void *status ;  
  
    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);  
  
    thread_args.db = db_open(DATAS);  
  
    start = get_ustime_sec();  
    for (i = 0; i < count; i++) {  
        if (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);  
  
        thread_args.sk.length = KSIZE;  
        thread_args.sk.mem = key;  
        thread_args.sv.length = VSIZE;  
        thread_args.sv.mem = val;  
        int k;  
        for (k=0; k<nimata; k++){  
            pthread_create(&tid[k],NULL,db_add,(void *) &thread_args); //create thread  
        }  
        for(k=0; k<nimata; k++){  
            pthread_join(tid[k],&status);  
        }  
    }  
}
```



```
void _read_test(long int count, int r, int nimata)
{
    pthread_t tid[nimata]; //create
    struct data thread_args; //struct
    int i;
    int found = 0;
    double cost;
    long long start, end;
    char key[KSIZE + 1];
    void *status;

    thread_args.db = db_open(DATAS);
    start = get_uptime_sec();
    for (i = 0; i < count; i++) {
        memset(key, 0, KSIZE + 1);

        /* if you want to test random write, use the following */
        if (r)
            _random_key(key, KSIZE);
        else
            snprintf(key, KSIZE, "key-%d", i);
        fprintf(stderr, "%d searching %s\n", i, key);
        thread_args.sk.length = KSIZE;
        thread_args.sk.mem = key;
        int k;
        for (k=0; k<nimata; k++){
            pthread_create(&tid[k], NULL, db_get, (void *) &thread_args); //create thread
        }
        for(k=0; k<nimata; k++){
            pthread_join(tid[k], &status);
        }
    }
}
```

Για να μπορέσουμε να χρησιμοποιήσουμε αυτήν την λειτουργία θα πρέπει στο Terminal να γραφουμε την παρακάτω εντολή (όπου 6 ο αριθμός των νημάτων).

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench read 100000 6
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench write 100000 6
```

Test με κατάλληλο μηνύματα ότι έφτιαξε όσα νήματα του ζητήσαμε.

```
|Random-Write (done:100000): 0.000150 sec/op; 6666.7 writes/sec(estimated); cost:15.000(sec);
we have create 6 threads
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
|Random-Read (done:100000, found:100000): 0.000100 sec/op; 10000.0 reads /sec(estimated); cost:10.000(sec)
we have create 6 threads
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Εισαγωγή νέας λειτουργίας *readwrite* όπου εκτελούνται παράλληλα *puts* και *gets* με βάση ένα ποσοστό που δίνεται ως παράμετρο στη γραμμή εντολών.

Για την εισαγωγή της νέας λειτουργίας *readwrite* όπου εκτελούνται παράλληλα *puts* και *gets* .Θα χρειαστεί να υλοποιήσουν πολυνοματικά την *db_add*.

Λειτουργία *db_add* πολυνοματικά

Με παρόμοιο τρόπο με τον παραπάνω καταφέραμε να δημιουργήσουμε και την *db_add* να λειτουργεί πολυνοματικά τροποποιώντας την *_write_test* .

- ☒ Ο κώδικας που έχουμε προσθέσει στο αρχείο *kiwi.c* φαίνεται παρακάτω με ***bold***

```
void _write_test(long int count, int r)
{
    pthread_t tid[5]; //create
    struct data thread_args; //struct
    int i;
    double cost;
    long long start,end;
    void *status ; //the return argument of join(instead of ret

    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);

    thread_args.db = db_open(DATAS); //arguments for db get forward to struct

    start = get_ustime_sec();
    for (i = 0; i < count; i++) {
        if (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);

thread_args.sk.length = KSIZE; //arguments for db get forward to struct
thread_args.sk.mem = key; //arguments for db get forward to struct
thread_args.sv.length = VSIZE; //arguments for db get forward to struct
thread_args.sv.mem = val; //arguments for db get forward to struct

int k;
for (k=0; k<5; k++){
    pthread_create(&tid[k],NULL,db_add,(void *) &thread_args); //create thread
}
for(k=0; k<5; k++){
    pthread_join(tid[k],&status); //waits for a thread to terminate
}
//db_add(db, &sk, &sv);
if ((i % 10000) == 0) {
    fprintf(stderr,"random write finished %d ops%30s\r",
            i,
            "");

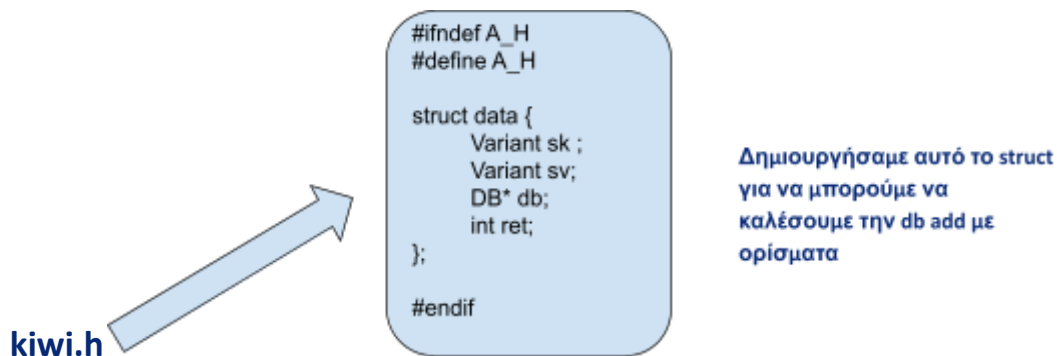
    fflush(stderr);
}
}

db_close(thread_args.db);//we use it for the struct

end = get_ustime_sec();
cost = end -start;

printf(LINE);
printf("| Random-Write      (done:%ld): %.6f sec/op; %.1f writes/sec(estimated);
cost:%.3f(sec);\n"
    ,count, (double)(cost / count)
    ,(double)(count / cost)
    ,cost);
}
```

Δημιουργούμε τα νηματα σύμφωνα με τις διαφάνειες που έχουμε δει στο μαθημα και χρησιμοποιούμε ένα struct στο αρχείο kiwi.h για να μπορούμε να περάσουμε τα ορίσματα στην db_add



- ☒ Ο κώδικας που έχουμε τροποποιήσει στο αρχείο db.c φαίνεται παρακάτω με **bold**

Χρησιμοποιήσαμε την δομή της διαφάνειας που επεξηγεί πώς να περάσουμε πολλές παραμέτρους στη συνάρτησή μας

```
void *db_add(void *arg)
{
    int x;
    struct data *d=(struct data *) arg;
    pthread_mutex_lock(&mutex);
    if (memtable_needs_compaction(d->db->memtable))
    {
        INFO("Starting compaction of the memtable after %d insertions and %d deletions",
            d->db->memtable->add_count, d->db->memtable->del_count);
        sst_merge(d->db->sst, d->db->memtable);
        memtable_reset(d->db->memtable);
    }
    x=memtable_add(d->db->memtable, &d->sk, &d->sv);
    pthread_mutex_unlock(&mutex);
    return (void *)(intptr_t)x;
}
```

Και έγιναν και οι δηλώσεις στην db.h

```
void *db_add(void *arg);
```

```
void *db_get(void *arg);
```

Test για την σωστή λειτουργία του db_add και bench χρόνου :

Το test που κάναμε για να διάγνωση της σωστής λειτουργίας του ποληνυματισμού είναι παρόμοια με αυτή που κάναμε στην db_get με μηνύματα ποτε ξεκινάνε και πότε σταματάνε τα νήματα

```
for (k=0; k<5; k++){
    pthread_create(&tid[k],NULL,db_add,(void *) &thread_args); //create thread
    printf("Thread %d has started\n",k);
}
for(k=0; k<5; k++){
    pthread_join(tid[k],&status);
    printf("Thread %d has finished\n",k);
}
```

```
Thread 0 has started
Thread 1 has started
Thread 2 has started
Thread 3 has started
Thread 4 has started
Thread 0 has finished
Thread 1 has finished
Thread 2 has finished
Thread 3 has finished
Thread 4 has finished
```

Παρατήρηση όπως φαίνεται και στο παραπάνω screenshot τα νήματα ξεκινάνε ταυτόχρονα και τερματίζουν ταυτόχρονα.

5 Threads bench:

```
[Random-Write (done:100000): 0.000120 sec/op; 8333.3 writes/sec(estimated); cost:12.000(sec);
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Αυτο τον χρόνο θα τον χρειαστούμε παρακατω οταν φτιαξουμε την readwrite για να κανουμε τα benchmark.

Δημιουργία `readwrite`

Η νέα λειτουργία `readwrite` όπου εκτελούνται παράλληλα `puts` και `gets` με βάση ένα ποσοστό που δίνεται ως παράμετρο στη γραμμή εντολών. Αν πχ το ποσοστό είναι 50 τότε έχουμε 50% PUTs και 50% GETs.

Σύνταξη της `readwrite`

`./kiwi-bench readwrite <count> <threads> <percentage for read>`

- Terminal input example

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 100000 8 50
```

- Terminal output example

```
|Random-Read      (done:100000, found:100000): 0.000050 sec/op; 20000.0 reads /sec(estimated); cost:5.000(sec)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Write     (done:100000): 0.000090 sec/op; 11111.1 writes/sec(estimated); writecost:9.000(sec);
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
we have 4 threads for read
we have 4 threads for write
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

Όπως είναι λογικό διαμερισμός του 50% των 8 νημάτων αντιστοιχεί σε 4 για την `read` και 4 για την `write`

Μερικά ακόμα παραδείγματα για την τεκμηρίωση της ορθής λειτουργίας.

- Terminal input example

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 100000 10 10
```

- Terminal output example

```
|Random-Read      (done:100000, found:100000): 0.000040 sec/op; 25000.0 reads /sec(estimated); cost:4.000(sec)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Write     (done:100000): 0.000270 sec/op; 3703.7 writes/sec(estimated); writecost:27.000(sec);
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
we have 1 threads for read
we have 9 threads for write
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

- Terminal input example

```
myy601@myy601lab1:~/kiwi/kiwi-source/bench$ ./kiwi-bench readwrite 100000 10 80
```

- Terminal output example

```
|Random-Read      (done:100000, found:100000): 0.000170 sec/op; 5882.4 reads /sec(estimated); cost:17.000(sec)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Random-Write     (done:100000): 0.000050 sec/op; 20000.0 writes/sec(estimated); writecost:5.000(sec);
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
we have 8 threads for read
we have 2 threads for write
myy601@myy601lab1:~/kiwi/kiwi-source/bench$
```

- ☒ Ο κώδικας που έχουμε τροποποιήσει στο αρχείο *bench.c* φαίνεται παρακάτω με **bold**

```
else if(strcmp(argv[1], "readwrite") == 0) {  
    int r = 0;  
    double pososto; //we use it to take the persentage to calculate how many threads to use.  
    int nimata; //we use it to take the number of threads from main arg  
    nimata=atoi(argv[3]); //take arg 3 as intenger  
    pososto=atof(argv[4]); //take arg 4 as float  
    count = atoi(argv[2]);  
    _print_header(count);  
    _print_environment();  
    if (argc == 6) //we change this from 5 to 6.  
        r = 1;  
    _readwrite(count,r,nimata,pososto); //new readwrite  
}
```

- ☒ Ο κώδικας που έχουμε τροποποιήσει στο αρχείο *kiwi.c* φαίνεται παρακάτω με **bold**

```
void _read_write_test(long int count, int r,int nimata_r,int nimata_w){  
  
    //initialalze read's variables  
  
    pthread_t tid[nimata_r]; //create 5 threads  
    struct data thread_args; //use struct in function  
    int i;  
    int found = 0;  
    double cost;  
    long long start,end;
```

```
char key[KSIZE + 1];
void *status ; //the variable for the return

thread_args.db = db_open(DATAS); //arguments for db get forward to struct
start = get_ustime_sec();

//inititalize write's variables

pthread_t tidwrite[nimata_w]; //create
double writecost;
long long writestart, writeend;

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

//code for read

start = get_ustime_sec();
for (i = 0; i < count; i++) {
    memset(key, 0, KSIZE + 1);

    /* if you want to test random write, use the following */
    if (r)
        _random_key(key, KSIZE);
    else
        snprintf(key, KSIZE, "key-%d", i);
    fprintf(stderr, "%d searching %s\n", i, key);
    thread_args.sk.length = KSIZE; //arguments for db get forward to struct
    thread_args.sk.mem = key; //arguments for db get forward to struct
    int k;
    for (k=0; k<nimata_r; k++){
        pthread_create(&tid[k], NULL, db_get, (void *) &thread_args); //create thread
    }
    for(k=0; k<nimata_r; k++){
        pthread_join(tid[k], &status); //releasing thread resources
    }
}
```



```
    }  
    if ((intptr_t)status) {//the return of db get  
        //db_free_data(sv.mem);  
        found++;  
    }  
  
    if ((i % 10000) == 0) {  
        fprintf(stderr, "random read finished %d ops%30s\r",  
                i,  
                "");  
  
        fflush(stderr);  
    }  
  
}  
  
end = get_ustime_sec();  
//-----  
//code for write  
  
writestart = get_ustime_sec();  
  
for (i = 0; i < count; i++) {  
    if (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);  
  
    thread_args.sk.length = KSIZE; //arguments for db get forward to struct  
    thread_args.sk.mem = key; //arguments for db get forward to struct  
    thread_args.sv.length = VSIZE; //arguments for db get forward to struct  
    thread_args.sv.mem = val; //arguments for db get forward to struct  
    int k;  
    for (k=0; k<nimata_w; k++){  
        pthread_create(&tidwrite[k], NULL, db_add, (void *) &thread_args);  
  
//create thread
```

```
    }  
    for(k=0; k<nimata_w; k++){  
        pthread_join(tidwrite[k],&status);//waits for a thread to terminate  
    }  
    //db_add(db, &sk, &sv);  
    if ((i % 10000) == 0) {  
        fprintf(stderr,"random write finished %d ops%30s\r",  
                i,  
                "");  
  
        fflush(stderr);  
    }  
}  
  
//-----  
//db close  
db_close(thread_args.db);//we use it for the struct  
  
cost = end -start;//read total time  
  
writeend = get_ustime_sec();//stop write stopwatch  
writecost=writeend-writestart;//write total time  
  
printf(LINE);  
printf("| Random-Read    (done:%ld, found:%d): %.6f sec/op; %.1f reads  
/sec(estimated); cost:%.3f(sec)\n",  
        count, found,  
        (double)(cost / count),  
        (double)(count / cost),  
        cost);  
  
printf(LINE);  
printf("| Random-Write    (done:%ld): %.6f sec/op; %.1f writes/sec(estimated);  
writecost:%.3f(sec);\n"  
        ,count, (double)(writecost / count)  
        ,(double)(count / writecost)  
        ,writecost);  
  
printf(LINE);
```

```
    printf("we have %d threads for read \n we have %d threads for  
write\n",nimata_r,nimata_w);  
}  
void _readwrite(int count,int r,int nimata,double pososto){  
^  int nimata_r;  
    int nimata_w;  
    if(pososto<49.0){  
        pososto=pososto-100;//take the difference  
        pososto=abs(pososto);//absolute the result  
        pososto = pososto/100;//make it %  
        nimata_w=pososto*nimata;//give the %  
        nimata_r=nimata-nimata_w;//give the %  
    }else{  
        pososto = pososto/100;//make it %  
        nimata_r=pososto*nimata;//give the %  
        nimata_w=nimata-nimata_r;//give the %  
    }  
    _read_write_test(count,r,nimata_r,nimata_w);}
```

Final Outputs

- Make all output

```
myy601@myy601lab1:~/kiwi/kiwi-source$ make all  
cd engine && make all  
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/engine'  
make[1]: Nothing to be done for 'all'.  
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/engine'  
cd bench && make all  
make[1]: Entering directory '/home/myy601/kiwi/kiwi-source/bench'  
gcc -g -ggdb -Wall -Wno-implicit-function-declaration -Wno-unused-but-set-variable bench.c kiwi.c -L ../engine -lindexer -lpthread -lsnappy -o kiwi-bench  
make[1]: Leaving directory '/home/myy601/kiwi/kiwi-source/bench'  
myy601@myy601lab1:~/kiwi/kiwi-source$
```

² Η bench.c καλεί την _readwrite και αυτή με την σειρά της καλεί την _read_write_test η οποία αποτελείται ουσιαστικά από τις _write και _read τροποποιημένης κατάλληλα