Λειτουργικά Συστήματα άσκηση 1

Παύλος Ορφανίδης, AM:4134 Χρήστος Νάσης, AM:3047

3 Απριλίου 2023

Περιεχόμενα

1 BENCH.C[1]

```c
//Kanoume aithsh gia eggrafh
void *write_request(void *arg)
{       struct data *d = (struct data *)arg;
        _write_test(d->myCount, d->my_r, d->myThreads);                 //Klhsh ths _write_test() tou kiwi.c,wste na ginei h eggrafh pou zhtame
        return 0;
}

//Kanoume aithsh gia anagnwsh
void *read_request(void *arg)
{       struct data *d = (struct data *)arg;
        _read_test(d->myCount, d->my_r, d->myThreads);                  //Klhsh ths _read_test() tou kiwi.c,wste na ginei h anagnwshh pou zhtame
        return 0;
}
```

Σχήμα 1: Οι συναρτήσεις write_request και read_request

```c
    int r = 0;
    if (argc < 4) {
            fprintf(stderr,"Usage: db-bench <write | read | readwrite> <count> <threads> <percentage>\n");         //Pros8esame thn periptwsh readwrite,to plh8os twn nhmatwn(threads),
to r kai to pososto(percentage) gia thn periptwsh readwrite
            exit(1);
    }
    int threads = atoi(argv[3]);           //Plh8os nhmatwn
    long int count;
    int i;
    pthread_t threadsGetID[threads], threadsPutID[threads]; //Pinakes tautothtwn(anagnwristikwn) nhmatwn pou exoun megethos iso me to plhthos twn nhmatwn
    struct data argsPut,argsGet;                    //Orismata twn sunarthsewn pthread_create()
    pthread_mutex_init(&totalPut,NULL);             //Arxikopoihsh kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes eggrafes
    pthread_mutex_init(&totalGet,NULL);             //Arxikopoihsh kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes anagnwseis
    costOfTotalPut = 0;                             //Arxikopoihsh sunolikou kostous(xronou) eggrafwn sto 0
    costOfTotalGet = 0;                             //Arxikopoihsh sunolikou kostous(xronou) anagnwsewn sto 0
    srand(time(NULL));
```

Σχήμα 2: Αρχικοποίηση των μεταβλητών μέσα στη συνάρτηση main()

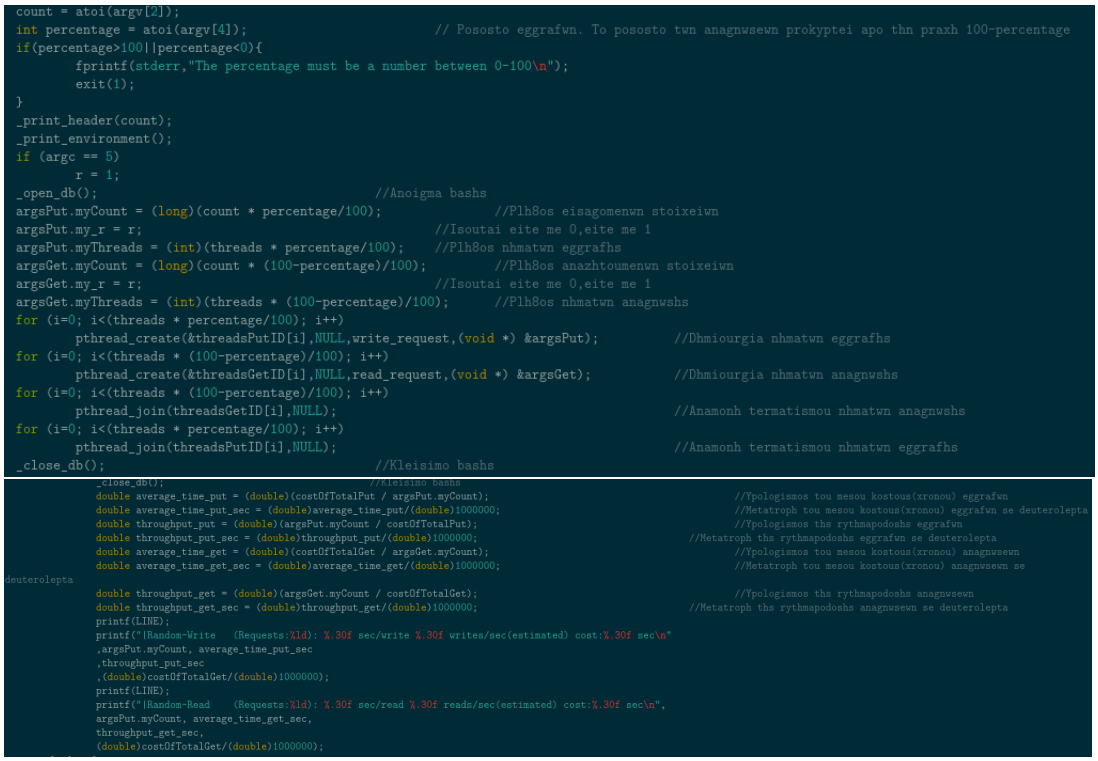
```c
    if (strcmp(argv[1], "write") == 0) {

            count = atoi(argv[2]);
            _print_header(count);
            _print_environment();
            if (argc == 4)
                    r = 1;
            _open_db();                             //Anoigma bashs
            argsPut.myCount = count;                        //Plh8os eisagomenwn stoixeiwn
            argsPut.my_r = r;                               //Isoutai eite me 0,eite me 1
            argsPut.myThreads = threads;            //Plh8os nhmatwn
            for (i=0; i<threads; i++)
                    pthread_create(&threadsPutID[i],NULL,write_request,(void *) &argsPut);          //Dhmiourgia nhmatwn
            for (i=0; i<threads; i++)
                    pthread_join(threadsPutID[i],NULL);                                             //Anamonh termatismou nhmatwn
            _close_db();                            //Kleisimo bashs
            double average_time = (double)(costOfTotalPut / argsPut.myCount);                               //Ypologismos tou mesou kostous(xronou) eggrafwn
            double average_time_sec = (double)average_time/(double)1000000;                                 //Metatroph tou mesou kostous(xronou) eggrafwn se deuterolepta
            double throughput = (double)(argsPut.myCount / costOfTotalPut);                         //Ypologismos ths rythmapodoshs eggrafwn
            double throughput_sec = (double)throughput/(double)1000000;                             //Metatroph ths rythmapodoshs eggrafwn se deuterolepta
            printf(LINE);
            printf("|Random-Write    (Requests:%ld): %.30f sec/write %.30f writes/sec(estimated) cost:%.30f sec\n"
            ,argsPut.myCount, average_time_sec
            ,throughput_sec
            ,(double)costOfTotalPut/(double)1000000);
```

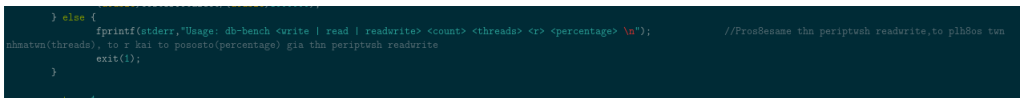
Σχήμα 3: Η περίπτωση όπου γίνεται αίτηση εγγραφής

[1] Φάκελος 'kiwi/kiwi-source/bench'

Σχήμα 4: Η περίπτωση όπου γίνεται αίτηση ανάγνωσης



Σχήμα 5: Η περίπτωση όπου γίνεται αίτηση ταυτόχρονης εγγραφής και ανάγνωσης



Σχήμα 6: Η περίπτωση όπου ο χρήστης ζητά λάθος λειτουργία

1.1 write_request(void *arg)

Αρχικά, προσθέσαμε πάνω από τη main() τη συνάρτηση void *write_request(void *arg), με την οποία κάνουμε αίτηση για εγγραφή (put). Η υλοποίηση της αιτού-

μενης εγγραφής γίνεται μόλις κληθεί η συνάρτηση _write_test() του αρχείου KIWI.C

1.2 read_request(void* arg)

Ακριβώς κάτω από τη συνάρτηση write_request(void *arg), ορίσαμε με ανάλογο τρόπο τη συνάρτηση void *read_request(void *arg), με την οποία κάνουμε αίτηση για ανάγνωση (get). Η υλοποίηση της αιτουμενης ανάγνωσης γίνεται μόλις κληθεί η συνάρτηση _read_test() του αρχείου KIWI.C.

1.3 main

Στη main() αρχικά αλλάξαμε τη συνθήκη στο if (argc $<$ 3) στη γραμμή 89, όπου στη θέση του 3 βάλαμε το 4. Αυτό το κάναμε, γιατί τα ορίσματα που πρέπει να δίνουμε στο τερματικό θέλουμε να είναι 4. Συγκεκριμένα, τα ορίσματα αυτά είναι:

1. το όνομα του εκτελέσιμου αρχείου ./kiwi-bench,
2. τη λειτουργία που θέλουμε να εκτελεστεί (write ή read),
3. το πλήθος των εισαγόμενων/αναζητούμενων στοιχείων στην αντίστοιχη λειτουργία και
4. το πλήθος των νημάτων.

Ιδιαίτερη περίπτωση αποτελεί η λειτουργία readwrite, όπου υλοποιείται ταυτόχρονα ανάγνωση και εγγραφή. Στην περίπτωση αυτή, πρέπει να δίνουμε στο τερματικό 5 ορίσματα, τα οποία είναι τα 4 ορίσματα που αναφέρθηκαν προηγουμένως και το 5ο όρισμα είναι το ποσοστό εγγραφών. Το ποσοστό αναγνώσεων προκύπτει από την πράξη: 100 - ποσοστό εγγραφών, όπου θα εξηγήσουμε παρακάτω. Αν τα ορίσματα που δίνουμε στο τερματικό είναι κάτω από 4, τότε δεν εκτελείται καμία λειτουργία και η πολυνηματική μηχανή τερματίζει. Μέσα στο if-block της γραμμής 89, στο fprintf της γραμμής 90 προσθέσαμε την περίπτωση readwrite, το πλήθος των νημάτων (threads) και το ποσοστό εγγραφών (percentage) για την περίπτωση readwrite. Το percentage ορίστηκε ώς ακέραιος για να επιστρέφει η διαίρεση με το count ακέραιο αριθμό. Σε αντίθετη περίπτωση θα επέστρεφε double με αποτέλεσμα το threads να είναι double και αυτό. Οπότε ο πίνακας που περιέχει τα threads θα αρχικοποιούνταν με double μέγεθος και θα έβγαζε σφάλμα. Έπειτα, προσθέσαμε τις ακέραιες μεταβλητές threads και i. Η μεταβλητή i χρησιμοποιείται ως δείκτης των πινάκων threadsPutID[threads] και threadsGetID[threads] που ορίζουμε παρακάτω κατά την δημιουργία (pthread_create()) και την αναμονή τερματισμού (pthread_join()) των νημάτων εγγραφής και ανάγνωσης αντίστοιχα, ενώ η μεταβλητη threads εκφράζει το πλήθος των νημάτων, το οποίο δίνουμε ώς όρισμα στο τερματικό (argv[3]). Καλό θα ήταν όμως να μην χρησιμοποιούμε πολλά νήματα, γιατί υπερφορτώνουμε τον επεξεργαστή με αποτέλεσμα το λειτουργικό σύστημα να δυσκολεύεται να διαχειριστεί τόσο μεγάλο αριθμό νημάτων. Στην συνέχεια δηλώσαμε τις μεταβλητές τύπου struct data argsPut και argsGet. Το argsPut χρησιμοποιείται στην περίπτωση που θέλουμε να γίνεται εγγραφή, ενώ το argsGet χρησιμοποιείται στην περίπτωση που θέλουμε να γίνεται ανάγνωση. Επίσης, τα argsPut και argsGet χρησιμοποιούνται κατά την δημιουργία

(pthread_create()) και κατά την αναμονή τερματισμού (pthread_join()) των νημάτων, επειδή όταν κάνουμε αίτηση για εγγραφή, ανάγνωση ή και τις 2 λειτουργίες ταυτόχρονα πρέπει οι τιμές τους να περνιούνται με αναφορα στις αντίστοιχες συναρτήσεις (*write_request() ή *read_request()), ώστε να εκτελεστεί το αντίστοιχο νήμα. Επιπλέον, μέσω αυτών αποκτουμε πρόσβαση στα πεδία της δομής struct data(myCount, my_r, myThreads), που είναι ορισμένη στο αρχείο BENCH.H και τους δίνουμε τις τιμές count, r και threads αντίστοιχα. Έπειτα, ορίσαμε τους πίνακες threadsPutID[threads] και threadsGetID[threads], στους οποίους αποθηκεύονται αντίστοιχα τα αναγνωριστικά των νημάτων εγγραφής και ανάγνωσης που θα δημιουργούνται. Το μέγεθος των 2 πινάκων θα είναι ίσο με το πλήθος των νημάτων threads που δίνουμε ως όρισμα στο τερματικό(argv[3]).

Στις γραμμές 98-99, χρησιμοποιούμε 2 κλειδαριές αμοιβαίου αποκλεισμού(mutexes), τις οποίες αρχικοποιήσαμε δυναμικά ως εξής:

- pthread_mutex_init(&totalPut,NULL);
- pthread_mutex_init(&totalGet,NULL);

Οι κλειδαριές αυτές χρησιμοποιούνται, ώστε να μην εκτελούνται ταυτόχρονα τα νήματα εγγραφής και ανάγνωσης στη μηχανή αποθήκευσης.

Στη συνέχεια, αρχικοποιήσαμε τους χρόνους (κόστη) των εγγραφών και των αναγνώσεων αντίστοιχα costOfTotalPut και costOfTotalGet (ο ορισμός τους έγινε στο BENCH.H) στο 0.

Μέσα στα block όπου ελέγχονται τα περιεχόμενα του argv[1] (αν είναι ίσο με "write", "read" ή "readwrite") αρχικά αλλάζει η τιμή του r από 0 σε 1, αν το πλήθος των ορισμάτων που δίνουμε στο τερματικό είναι 4 (read ή write) ή 5 (readwrite). Ακολούθως, ανοίγουμε τη βάση db του αρχείου KIWI.C, έτσι ώστε να μπορέσουμε να δημιουργήσουμε τα νήματα. Αφού την ανοίξουμε, δημιουργούμε τα νήματα με τη συνάρτηση pthread_create(), εκτελώντας τόσες επαναλήψεις, όσο και το πλήθος των νημάτων (threads). Στην περίπτωση που υλοποιούμε ταυτόχρονη εγγραφή και ανάγνωση (readwrite), πρέπει να προσδιορίσουμε το ποσοστό από τον κάθε τύπο (put και get). Το ποσοστό εγγραφών (percentage) το ορίζουμε μέσα στο αντίστοιχο if-block και η τιμή του προσδιορίζεται από την παράμετρο argv[4]. Αντίστοιχα, το ποσοστό αναγνώσεων προκύπτει από την πράξη: 100 - percentage. Η τιμή του percentage πρέπει να παίρνει τιμές μεταξύ του 0 και του 100. Αν ο χρήστης πληκτρολογήσει μια τιμή που βρίσκεται εκτός του πεδίου τιμών του percentage, τότε τυπώνεται μήνυμα σφάλματος και η πολυνηματική μηχανή τερματίζει. Αφού δημιουργηθούν όλα τα νήματα (πρώτα δημιουργούνται τα νήματα εγγραφής στην περίπτωση που υλοποιούμε readwrite),πρέπει να περιμένουμε τον τερματισμό τους. Αυτό το κάνουμε με τη συνάρτηση pthread_join(). Ακολούθως, κλείνουμε τη βάση db του αρχείου KIWI.C. Μετά το κλείσιμο της βάσης, υπολογίζουμε το συνολικό μέσο κόστος ανά εγγραφή/ανάγνωση(average_time) και το πλήθος των εγγραφών/αναγνώσεων που υλοποιούνται ανά κόστος ή αλλιώς ρυθμαπόδοση (rev_average_time), τα οποία μετατρέπουμε στη συνέχεια σε δευτερόλεπτα, διαιρώντας τα με το 1000000 (average_time_sec, rev_average_time_sec). Τέλος, μεταφέραμε τις 2 τελευταίες εντολές (printf) των συναρτήσεων _write_test() και _read_test() του αρχείου KIWI.C στο τέλος των blocks που ελέγχουμε την λειτουργία που εκτελούμε (read, write ή readwrite). Στην περίπτωση του readwrite, τοποθετούμε και τα 4 printf (2 για εγγραφή και 2 για ανάγνωση). Αυτά που θέλουμε να τυπώνονται σε κάθε περίπτωση είναι το πλήθος των εισαγόμενων/αναζητούμενων στοιχείων, το συνολικό μέσο κόστος ανά εγγραφή/ανάγνωση

σε δευτερόλεπτα(average_time_sec), το πλήθος των εγγραφών/αναγνώσεων που υλοποιούνται ανά δευτερόλεπτο ή αλλιώς ρυθμαπόδοση (rev_average_time_sec) και το συνολικό κόστος εγγραφών/αναγνώσεων εκφρασμένο σε δευτερόλεπτα.

2 BENCH.H[1]

```
extern pthread_mutex_t totalPut;        //Orismos kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes eggrafes
extern pthread_mutex_t totalGet;        //Orismos kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes anagnwseis
extern double costOfTotalPut;           //Sunoliko kostos(xronos) eggrafwn
extern double costOfTotalGet;           //Sunoliko kostos(xronos) anagnwsewn

extern int threads;                     //Orismos tou plh8ous twn nhmatwn
```

Σχήμα 7: Χρήση extern για να μην ορίζονται οι μεταβλητές 2 φορές

```
void _write_test(long int count, int r, int threads);          //Prwtotupo sunarthshs _write_test() tou arxeiou kiwi.c
void _read_test(long int count, int r, int threads);                   //Prwtotupo sunarthshs _read_test() tou arxeiou kiwi.c

void _open_db();                                                       //Prwtotupo sunarthshs _open_db() tou arxeiou kiwi.c
void _close_db();                                                      //Prwtotupo sunarthshs _close_db() tou arxeiou kiwi.c

//Orisame auth th domh, wste na apoktoume prosbash se authn mesw twn orismatwn args,args1 kai args2 tou bench.c
struct data
{
        long int myCount;               //Plh8os eisagomenwn/anazhtoumenwn stoixeiwn
        int my_r;                       //Isoutai eite me 0,eite me 1
        int myThreads;                  //Plh8os nhmatwn
};
```

Σχήμα 8: Αλλαγές στο BENCH.H

1. Δηλώσαμε ως extern τις παρακάτω κλειδαριές αμοιβαίου αποκλεισμού-mutexes (ο ορισμός τους γίνεται στο αρχείο KIWI.C):

   (α′) totalPut

   (β′) totalGet

   Οι κλειδαριές αυτές χρησιμοποιούνται αντίστοιχα για τις συνολικές εγγραφές και τις συνολικές αναγνώσεις που γίνονται στη μηχανή αποθήκευσης

2. Δηλώσαμε ως extern τους χρόνους (κόστη) των εγγραφών και των αναγνώσεων αντίστοιχα costOfTotalPut και costOfTotalGet (ο ορισμός τους γίνεται στο αρχείο KIWI.C)

3. Δηλώσαμε ως extern το πλήθος των νημάτων threads (ο ορισμός τους γίνεται στο αρχείο BENCH.C)

4. Ορίσαμε τα πρωτότυπα των συναρτήσεων _write_test(), _read_test(), _open_db() και _close_db() του αρχείου KIWI.C

5. Δημιουργήσαμε τη δομή struct data με τα εξής πεδία:

   (α′) *long int myCount*: πλήθος εισαγόμενων/αναζητούμενων στοιχείων

   (β′) *int my_r*: είναι σαν μια boolean μεταβλητή της java, δηλαδή ίση με 0 ή 1

   (γ′) *int myThreads*: πλήθος νημάτων

Η δομή αυτή δημιουργήθηκε, ώστε να μπορούμε να αποκτούμε σε αυτήν πρόσβαση μέσω των ορισμάτων argsPut και argsGet του αρχείου BENCH.C, όταν θέλουμε να δημιουργήσουμε τα νήματα ή όταν θέλουμε να περιμένουμε των τερματισμό τους

3 KIWI.C[1]

```
pthread_mutex_t totalPut;               //Orismos kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes eggrafes
pthread_mutex_t totalGet;               //Orismos kleidarias amoibaiou apokleismou(mutex) gia tis sunolikes anagnwseis
double costOfTotalPut;          //Sunoliko kostos(xronos) eggrafwn
double costOfTotalGet;          //Sunoliko kostos(xronos) anagnwsewn
```

Σχήμα 9: Ορισμός των μεταβλητών και των κλειδαριών αμοιβαίου αποκλεισμού (mutexes) για τις συνολικές εγγραφές και αναγνώσεις.

```
DB* db;                         //Orismos ths bashs ws global metablhth kai oxi entos twn sunarthsewn _write_test() kai _read_test()

//Anoigma bashs
void _open_db()
{
        db = db_open(DATAS);            //Klhsh ths sunarthshs db_open() tou db.c
}

//Kleisimo bashs
void _close_db()
{       db_close(db);                   //Klhsh ths sunarthshs db_close() tou db.c
}
```

Σχήμα 10: Οι συναρτήσεις που ανοίγουν και κλείνουν τη βάση αντίστοιχα

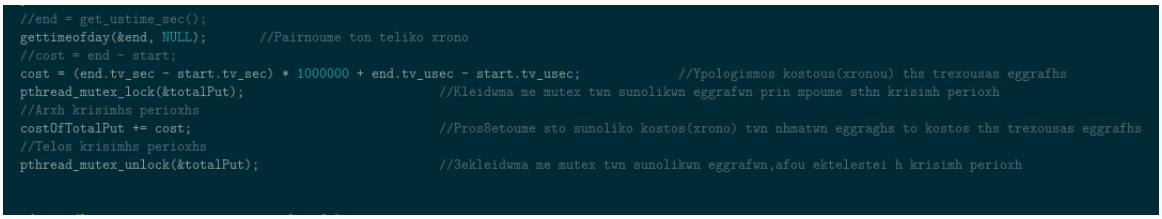
```
    long int newCount = count/threads;              //Neo plh8os eisagomenwn stoixeiwn kai diamoirasmos eggrafwn sta nhmata
    struct timeval start,end;       //Apo long long ta orisame ws struct timeval
```

Σχήμα 11: Ορισμός και αρχικοποίηση του πλήθους των αιτήσεων εγγραφής/ανάγνωσης και ορισμός αρχικού και τελικού κόστους (χρόνου) της τρέχουσας εγγραφής/ανάγνωσης

```
//start = get_usecs_sec();
gettimeofday(&start, NULL);     //Pairnoume ton arxiko xrono
for (i=0; i<newCount; i++) {
```

Σχήμα 12: Επιστροφή του αρχικού χρόνου (κόστους) της τρέχουσας εγγραφής/ανάγνωσης

Αρχικά, ορίσαμε 2 κλειδαριές αμοιβαίου αποκλεισμού (mutexes), χρησιμοποιούνται αντίστοιχα για τις συνολικές εγγραφές και τις συνολικές αναγνώσεις που γίνονται στη μηχανή αποθήκευσης. Έπειτα, ορίσαμε τη βάση db ως global και όχι εντός των συναρτήσεων _write_test() και _read_test(). Όταν θέλουμε να ανοίξουμε ή να κλείσουμε τη βάση db, καλούμε αντίστοιχα τις συναρτήσεις _open_db() ή _close_db(), τις οποίες δημιουργήσαμε εμείς. Στις συναρτήσεις _write_test() και _read_test() βάλαμε ως 3η παράμετρο το πλήθος των νημάτων (threads), ορίσαμε τη long int μεταβλητή newCount, την οποία θα χρειαστούμε, ώστε να μοιράζουμε στα νήματα τις εγγραφές (put) και τις αναγνώσεις (get) αντίστοιχα. Ο διαμοιρασμός αυτός θα γίνεται, διαιρώντας το πλήθος των εισαγόμενων/αναζητούμενων

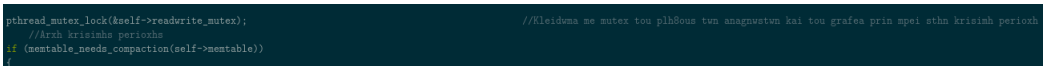


Σχήμα 13: Χρήση της κλειδαριάς αμοιβαίου αποκλεισμού, επιστροφή του τελικού χρόνου (κόστους) της τρέχουσας εγγραφής/ανάγνωσης και καθορισμός του συνολικού κόστους εγγραφών/αναγνώσεων

στοιχείων (count) με το πλήθος των νημάτων (threads). Επίσης, αλλάξαμε τον τύπο των μεταβλητών start και end. Συγκεκριμένα, αντί για long long, τα ορίσαμε ως struct timeval, ώστε να παίρνουμε τους αντίστοιχους χρόνους από τη συνάρτηση gettimeofday() και όχι από τη συνάρτηση get_ustime_sec(). Στα for-loops αλλάξαμε τη συνθήκη, βάζοντας newCount στη θέση του count, διότι έχει προηγηθεί διαμοιρασμός των εισαγόμενων/αναζητούμενων στοιχείων. Έπειτα, υπολογίσαμε το αντίστοιχο κόστος της τρέχουσας εγγραφής/ανάγνωσης ως εξής: $cost = (end.tv\_sec - start.tv\_sec) * 1000000 + end.tv\_usec - start.tv\_usec$; και όχι με την εντολή $cost = end–start$; Στη συνάρτηση _write_test(), αφού υπολογίσουμε το κόστος της τρέχουσας εγγραφής, κλειδώνουμε με mutex το σύνολο των εγγραφών που έχουν πραγματοποιηθεί μέχρι στιγμής, ώστε να προσθέτουμε στο συνολικό κόστος των εγγραφών το κόστος της τρέχουσας εγγραφής και μετά ξεκλειδώνουμε με mutex τις συνολικές εγγραφές. Αντίστοιχα, στη συνάρτηση _read_test(), αφού υπολογίσουμε το κόστος της τρέχουσας ανάγνωσης, κλειδώνουμε με mutex το σύνολο των αναγνώσεων που έχουν πραγματοποιηθεί μέχρι στιγμής, ώστε να προσθέτουμε στο συνολικό κόστος των αναγνώσεων το κόστος της τρέχουσας ανάγνωσης και μετά ξεκλειδώνουμε με mutex τις συνολικές αναγνώσεις. Τέλος, αφαιρέσαμε από τις 2 αυτές συναρτήσεις τις 2 τελευταίες εντολές τους (printf) και τις μεταφέραμε στο BENCH.C στα τμήματα κώδικα που κάνουμε εγγραφή (put) και ανάγνωση (get) αντίστοιχα. Στην περίπτωση του readwrite στο BENCH.C τοποθετούμε στο τέλος του block και τα 4 printf (2 για εγγραφή και 2 για ανάγνωση).

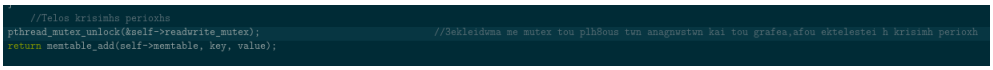
4 DB.C[2]

Σχήμα 14: Αρχικοποίηση των κλειδαριών για τους αναγνώστες και τον γραφέα

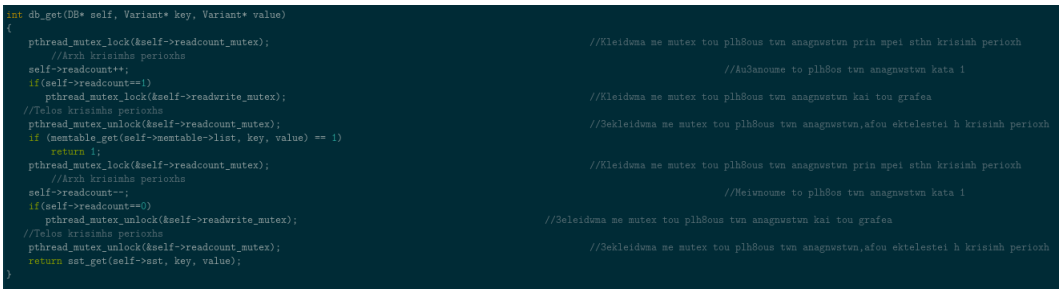


Σχήμα 15: Κλείδωμα πριν από ταυτόχρονη εγγραφή και ανάγνωση

[2]Φάκελος 'kiwi/kiwi-source/engine'



Σχήμα 16: Ξεκλείδωμα μετά από ταυτόχρονη εγγραφή και ανάγνωση



Σχήμα 17: Συνάρτηση ανάγνωσης από την βάση

Στη συνάρτηση db_open_ex() χρησιμοποιήσαμε 2 κλειδαριές αμοιβαίου αποκλεισμού (mutexes), τις οποίες αρχικοποιήσαμε δυναμικά ως εξής:
pthread_mutex_init(&(self→readwrite_mutex),NULL);
pthread_mutex_init(&(self→readcount_mutex),NULL);
Οι κλειδαριές αυτές χρησιμοποιούνται, ώστε να μην εκτελούνται ταυτόχρονα τα νήματα εγγραφής και ανάγνωσης στη μηχανή αποθήκευσης. Έπειτα, αρχικοποιήσαμε μέσω του self το readcount (ορισμένο στο DB.H) στο 0. Η μεταβλητή readcount εκφράζει το πλήθος των αναγνωστών. Στη συνάρτηση db_add(), η οποία είναι φτιαγμένη αποκλειστικά για εγγραφές (put), κλειδώνουμε με mutex το readwrite_mutex(ορισμένο στο DB.H), ώστε να μην γίνονται ταυτόχρονα εγγραφές(put) και αναγνώσεις(get). Δηλαδή, η επόμενη λειτουργία θα υλοποιηθεί αφού ολοκληρωθεί η προηγούμενη. Ακολούθως, ελέγχεται αν το log είναι γεμάτο, διότι στην περίπτωση αυτή δεν πρέπει να δέχεται νέες εγγραφές το memtable, αλλά πρέπει να γίνεται μαρκάρισμα του log, ώστε αυτό να συγχωνεύεται με τα sst αρχεία του level 0 ή του level 1 και να δημιουργείται νέο memtable που θα λαμβάνει τα νέα δεδομένα για εγγραφή(put). Έτσι, ενεργοποιείται η σύμπτυξη αρχείων(compaction). Να υπενθυμίσουμε ότι μόνο ένας γραφέας μπορεί να υλοποιεί την εγγραφή. Τέλος, πριν επιστρέψουμε το αλλαγμένο από τον γραφέα memtable, ξεκλειδώνουμε με mutex το readwrite_mutex. Αντίστοιχα, στη συνάρτηση db_get(), η οποία είναι φτιαγμένη αποκλειστικά για αναγνώσεις (get), κλειδώνουμε με mutex το readcount_mutex(ορισμένο στο DB.H), ώστε να μην μπαίνουν ταυτόχρονα πολλοί αναγνώστες. Δηλαδή, η επόμενη ανάγνωση θα υλοποιηθεί αφού ολοκληρωθεί η προηγούμενη. Έπειτα,αυξάνουμε το readcount (πλήθος αναγνωστών) κατά 1. Αν έχουμε μόνο έναν αναγνώστη που κάνει get, τότε πρέπει να κλειδώσουμε με mutex το readwrite_mutex, ώστε να μην γίνεται συγχρόνως και εγγραφή από κάποιον γραφέα. Ακολούθως, ξεκλειδώνουμε με mutex το readcount_mutex. Έπειτα, ψάχνουμε αν βρέθηκε το κλειδί(key) στο memtable. Αν βρέθηκε, τότε βγαίνουμε από τη συνάρτηση. Αλλιώς, ξανακλειδώνουμε με mutex το readcount_mutex, ώστε να μην μπαίνουν ταυτόχρονα πολλοί αναγνώστες. Έπειτα, μειώνουμε το readcount (πλήθος αναγνωστών) κατά 1. Αν δεν έχουμε κανέναν αναγνώστη που κάνει get, τότε πρέπει να ξεκλειδώσουμε με mutex το readwrite_mutex. Τέλος, ξεκλειδώνουμε με mutex το readcount_mutex.

5 DB.H[2]

```
typedef struct _db {
//    char basedir[MAX_FILENAME];
    char basedir[MAX_FILENAME+1];
    SST* sst;
    MemTable* memtable;
    pthread_mutex_t readwrite_mutex;                    //Orismos kleidarias amoibaiou apokleismou(mutex) gia tous anagnwstes kai ton grafea
    pthread_mutex_t readcount_mutex;                    //Orismos kleidarias amoibaiou apokleismou(mutex) mono gia tous anagnwstes
    int readcount;                                      //Plh8os anagnwstwn
} DB;
```

Σχήμα 18: Struct DB για ευκολότερη διαχείριση της βάσης

Μέσα στη δομή DB ορίσαμε 2 κλειδαριές αμοιβαίου αποκλεισμού (mutexes): το readwrite_mutex και το readcount_mutex, τις οποίες αρχικοποιούμε δυναμικά στο DB.C. Το readwrite_mutex χρησιμοποιείται όταν έχουν πρόσβαση στη βάση παράλληλα ένας ή περισσότεροι αναγνώστες και ένας γραφέας και το readcount_mutex χρησιμοποιείται όταν έχουν πρόσβαση στη βάση μόνο αναγνώστες. Έπειτα, ορίσαμε την ακέραια μεταβλητή readcount, με την οποία μετράμε το πλήθος των αναγνωστών (εφόσον επιτρέπεται η λειτουργία πολλαπλών αναγνώσεων) που μπαίνουν στην βαση.

6 Πίνακες και Διαγράμματα Στατιστικών της Πολυνηματικής Μηχανής kiwi-engine

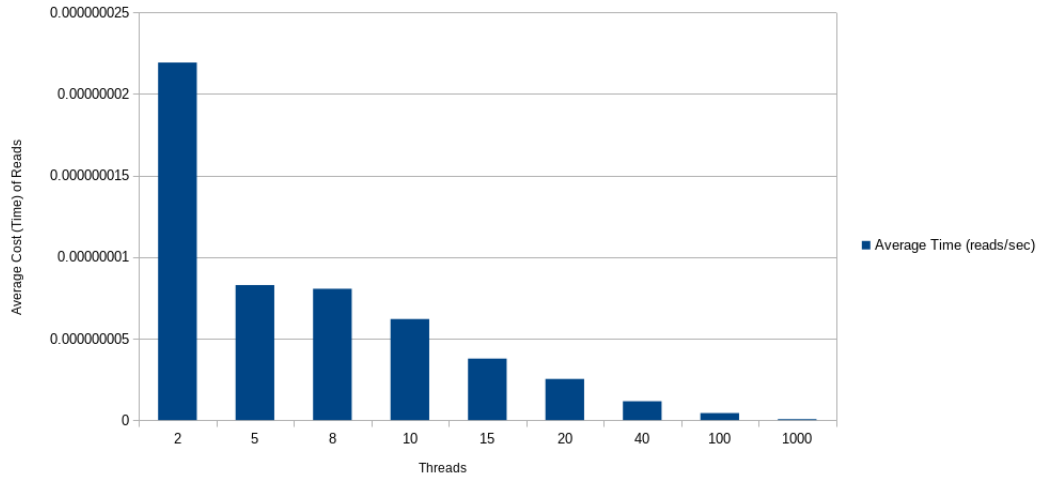
6.1 Περίπτωση Read

Count(requests)	Threads	Average Time (reads/sec)	Throughput (sec/read)	Cost (sec)
5	2	2.19298245614E-08	0.0000456	0.000228
20	5	8.28157349896E-09	0.00012075	0.002415
50	8	8.05152979066E-09	0.0001242	0.00621
100	10	6.19655471558E-09	0.00016138	0.016138
200	15	3.77081015856E-09	0.000265195	0.053039
500	20	2.5215338995E-09	0.000396584	0.198292
1000	40	1.15415200413E-09	0.000866437	0.866437
5000	100	4.3078897018E-10	0.0023213222	11.606611
10000	1000	5.664860968E-11	0.0176526839	176.526839

Σχήμα 19: Πίνακας Στατιστικών

Threads	Average Time (reads/sec)
2	2.19298245614E-08
5	8.28157349896E-09
8	8.05152979066E-09
10	6.19655471558E-09
15	3.77081015856E-09
20	2.5215338995E-09
40	1.15415200413E-09
100	4.3078897018E-10
1000	5.664860968E-11

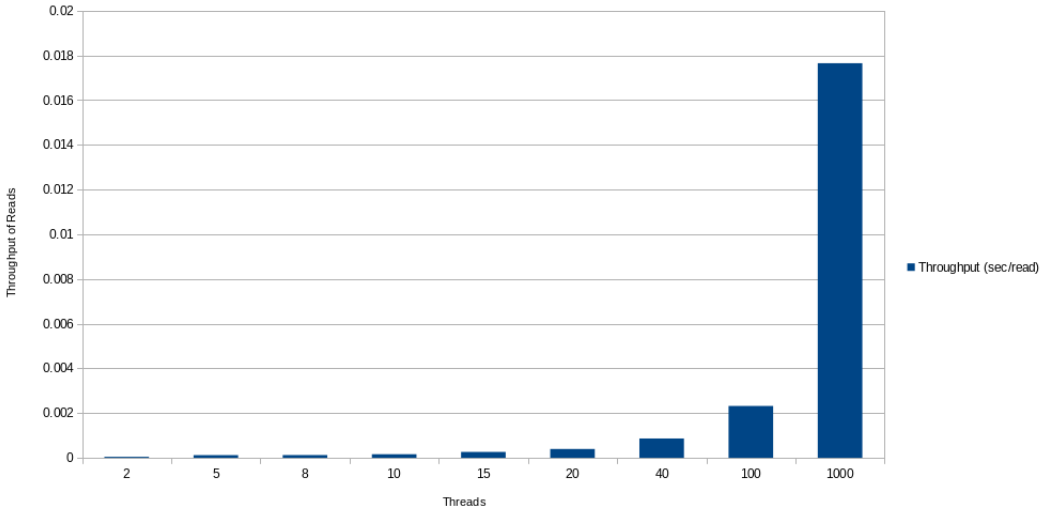
Σχήμα 20: Πίνακας Αποτελεσμάτων για το Μέσο Κόστος Αναγνώσεων



Σχήμα 21: Διάγραμμα Αποτελεσμάτων για το Μέσο Κόστος Αναγνώσεων

Threads	Throughput (sec/read)
2	0.0000456
5	0.00012075
8	0.0001242
10	0.00016138
15	0.000265195
20	0.000396584
40	0.000866437
100	0.0023213222
1000	0.0176526839

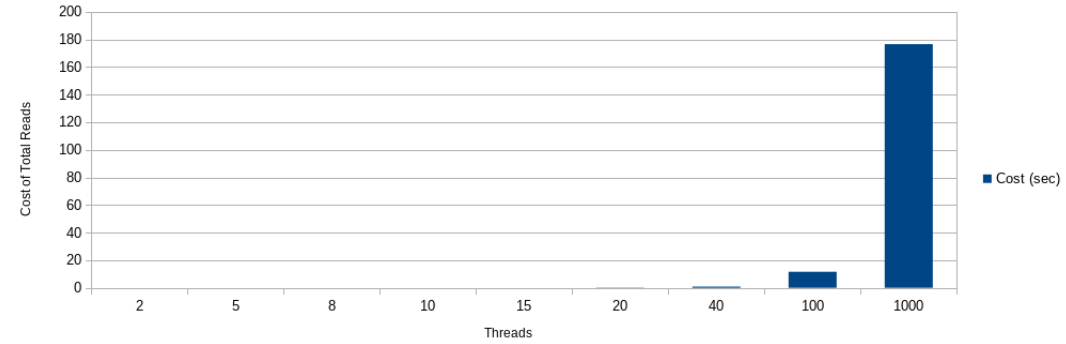
Σχήμα 22: Πίνακας Αποτελεσμάτων για την Ρυθμαπόδοση ανά Ανάγνωση



Σχήμα 23: Διάγραμμα Αποτελεσμάτων για την Ρυθμαπόδοση ανά Ανάγνωση

Threads	Cost (sec)
2	0.000228
5	0.002415
8	0.00621
10	0.016138
15	0.053039
20	0.198292
40	0.866437
100	11.606611
1000	176.526839

Σχήμα 24: Πίνακας Αποτελεσμάτων Κόστους Συνολικών Ανανώσεων



Σχήμα 25: Διάγραμμα Αποτελεσμάτων Κόστους Συνολικών Ανανώσεων

Παρατηρούμε ότι στο διάγραμμα 25 όταν το συνολικό κόστος είναι πολύ μικρό(απειροελάχιστο), το ορθογώνιο παραλληλόγραμμο δεν μπορεί να γίνει ορατό με γυμνό μάτι.

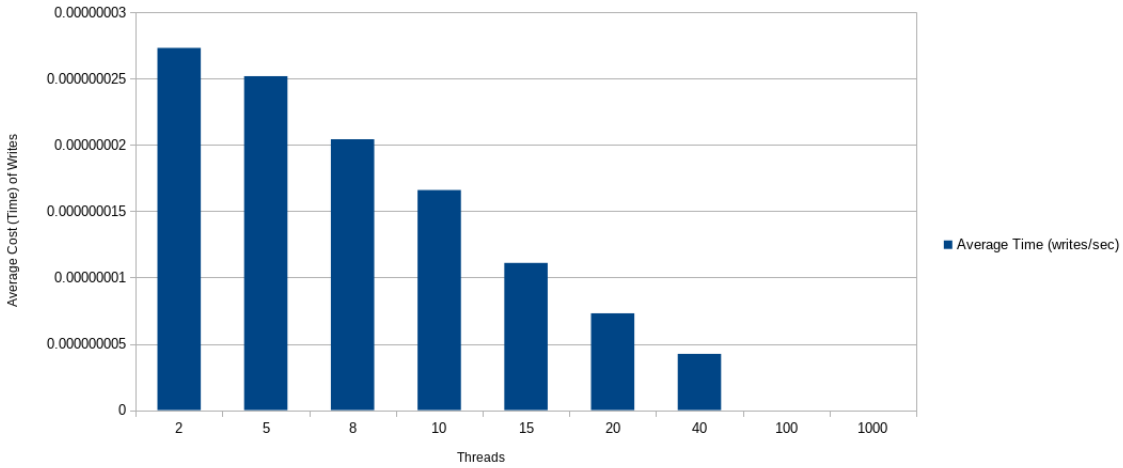
6.2 Περίπτωση Write

Count(requests)	Threads	Average Time (writes/sec)	Throughput (sec/write)	Cost (sec)
5	2	2.732240437158E-08	0.0000366	0.000183
20	5	2.518891687657E-08	0.0000397	0.000794
50	8	2.043318348999E-08	0.00004894	0.002447
100	10	1.660302174996E-08	0.00006023	0.006023
200	15	1.110124333925E-08	0.00009008	0.018016
500	20	7.30129524978E-09	0.000136962	0.068481
1000	40	4.24070226029E-09	0.00023581	0.23581
5000	100	Segmentation fault (core dumped)		
10000	1000	Segmentation fault (core dumped)		

Σχήμα 26: Πίνακας Στατιστικών

Threads	Average Time (writes/sec)
2	2.732240437158E-08
5	2.518891687657E-08
8	2.043318348999E-08
10	1.660302174996E-08
15	1.110124333925E-08
20	7.30129524978E-09
40	4.24070226029E-09
100	Segmentation fault (core dumped)
1000	Segmentation fault (core dumped)

Σχήμα 27: Πίνακας Αποτελεσμάτων για το Μέσο Κόστος Εγγραφών

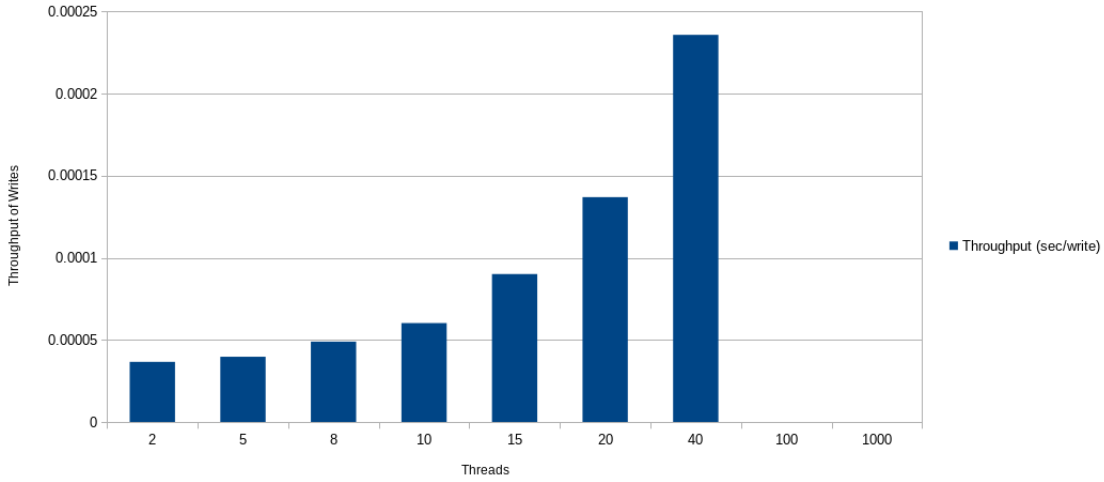


Σχήμα 28: Διάγραμμα Αποτελεσμάτων για το Μέσο Κόστος Εγγραφών

Παρατηρούμε ότι στο διάγραμμα 28 όταν κάνουμε πολλές αιτήσεις εγγραφής και χρησιμοποιούμε πολλά νήματα τότε γεμίζει η προσωρινή μνήμη του υπολογιστή, με αποτέλεσμα να μην μπορεί να αποθηκεύσει άλλα δεδομένα(Segmentation fault (core dumped)). Επομένως, στην περίπτωση αυτή δεν υπάρχει ορθογώνιο παραλληλόγραμμο.

Threads	Throughput (sec/write)
2	0.0000366
5	0.0000397
8	0.00004894
10	0.00006023
15	0.00009008
20	0.000136962
40	0.00023581
100	Segmentation fault (core dumped)
1000	Segmentation fault (core dumped)

Σχήμα 29: Πίνακας Αποτελεσμάτων για την Ρυθμαπόδοση ανά Εγγραφή

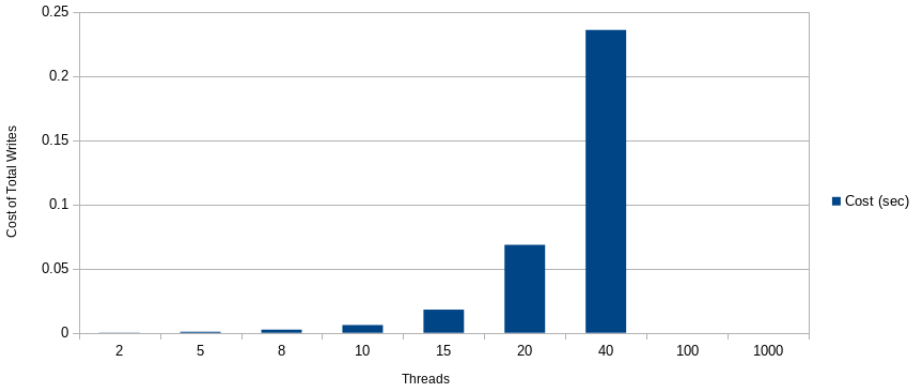


Σχήμα 30: Διάγραμμα Αποτελεσμάτων για την Ρυθμαπόδοση ανά Εγγραφή

Παρατηρούμε ότι στο διάγραμμα 30 όταν κάνουμε πολλές αιτήσεις εγγραφής και χρησιμοποιούμε πολλά νήματα τότε γεμίζει η προσωρινή μνήμη του υπολογιστή, με αποτέλεσμα να μην μπορεί να αποθηκεύσει άλλα δεδομένα(Segmentation fault (core dumped)). Επομένως, στην περίπτωση αυτή δεν υπάρχει ορθογώνιο παραλληλόγραμμο.

Threads	Cost (sec)
2	0.000183
5	0.000794
8	0.002447
10	0.006023
15	0.018016
20	0.068481
40	0.23581
100	Segmentation fault (core dumped)
1000	Segmentation fault (core dumped)

Σχήμα 31: Πίνακας Αποτελεσμάτων Κόστους Συνολικών Εγγραφών



Σχήμα 32: Διάγραμμα Αποτελεσμάτων Κόστους Συνολικών Εγγραφών

Παρατηρούμε ότι στο διάγραμμα 32 όταν το συνολικό κόστος είναι πολύ μικρό(απειροελάχιστο), το ορθογώνιο παραλληλόγραμμο δεν μπορεί να γίνει ορατό με γυμνό μάτι. Επίσης, όταν κάνουμε πολλές αιτήσεις εγγραφής και χρησιμοποιούμε πολλά νήματα τότε γεμίζει η προσωρινή μνήμη του υπολογιστή, με αποτέλεσμα να μην μπορεί να αποθηκεύσει άλλα δεδομένα(Segmentation fault (core dumped)). Επομένως, στην περίπτωση αυτή δεν υπάρχει ορθογώνιο παραλληλόγραμμο.

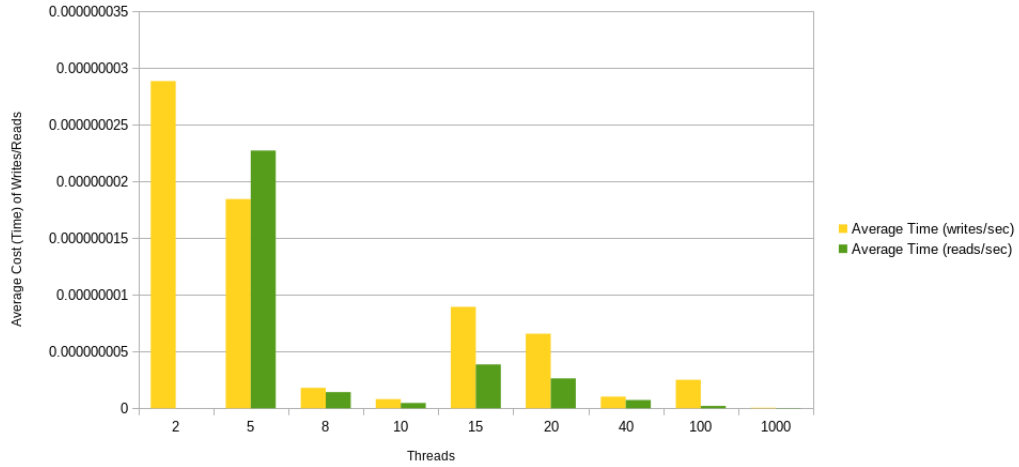
6.3 Περίπτωση ReadWrite

Count(requests)	Threads	percentage write	percentage read	Average Time (writes/sec)	Throughput (sec/write)	Cost Write (sec)	Average Time (reads/sec)	Throughput (sec/read)	Cost Read (sec)
5	2	73	27	2.88461538461539E-08	3.46666666666667E-05	0.000104	inf	0	0
20	5	50	50	1.8450184501845E-08	0.0000542	0.000542	2.27272727272727E-08	0.000044	0.00044
50	8	60	40	1.80136904047076E-09	0.000555133333333333	0.016654	1.42531356898518E-09	0.0007016	0.014032
100	10	30	70	8.07841447651874E-10	0.00123786666666667	0.037136	4.68252481738153E-10	0.0021356	0.149492
200	15	42	58	8.94378194207837E-09	0.000111809523809524	0.009392	3.86718229097213E-09	0.00025858620689655	0.029996
500	20	37	63	6.57473878740493E-09	0.000152097297297297	0.028138	2.63797001926137E-09	0.00037907936507937	0.11941
1000	40	90	10	1.02926897886225E-09	0.00097156333333333334	0.874407	7.28539060621735E-10	0.00137261	0.137261
5000	100	2	98	2.51268907985326E-09	0.00039798	0.039798	2.09377800962215E-10	0.00477605551020408	23.402672
10000	1000	20	80	4.92562309132105E-11	0.020302	40.604	2.39660231532814E-11	0.041725737875	333.805903

Σχήμα 33: Πίνακας Στατιστικών

Threads	percentage write	percentage read	Average Time (writes/sec)	Average Time (reads/sec)
2	73	27	2.88461538461539E-08	inf
5	50	50	1.8450184501845E-08	2.27272727272727E-08
8	60	40	1.80136904047076E-09	1.42531356898518E-09
10	30	70	8.07841447651874E-10	4.68252481738153E-10
15	42	58	8.94378194207837E-09	3.86718229097213E-09
20	37	63	6.57473878740493E-09	2.63797001926137E-09
40	90	10	1.02926897886225E-09	7.28539060621735E-10
100	2	98	2.51268907985326E-09	2.09377800962215E-10
1000	20	80	4.92562309132105E-11	2.39660231532814E-11

Σχήμα 34: Πίνακας Αποτελεσμάτων για το Μέσο Κόστος Ταυτόχρονων Αναγνώσεων και Εγγραφών

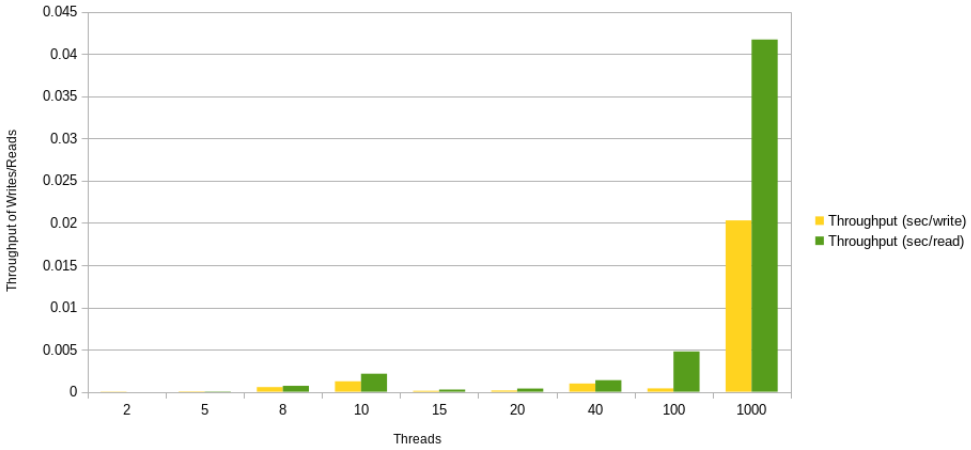


Σχήμα 35: Διάγραμμα Αποτελεσμάτων για το Μέσο Κόστος Ταυτόχρονων Αναγνώσεων και Εγγραφών

Παρατηρούμε ότι στο διάγραμμα 35 όταν το μέσο κόστος είναι πολύ μικρό(απειροελάχιστο), το ορθογώνιο παραλληλόγραμμο δεν μπορεί να γίνει ορατό με γυμνό μάτι. Επίσης, όταν το μέσο κόστος είναι άπειρο(inf), ενώ κανονικά θα έπρεπε να υπήρχε ορθογώνιο παραλληλόγραμμο που το ύψος του είναι άπειρο, δεν υπάρχει καθόλου. Αυτό συμβαίνει γιατί κατά την δημιουργία του διαγράμματος μέσω excel το inf αναγνωρίστηκε ώς string και όχι ώς αριθμός.

Threads	percentage write	percentage read	Throughput (sec/write)	Throughput (sec/read)
2	73	27	3.46666666666667E-05	0
5	50	50	0.0000542	0.000044
8	60	40	0.000555133333333333	0.0007016
10	30	70	0.00123786666666667	0.0021356
15	42	58	0.000111809523809524	0.000258586206896552
20	37	63	0.000152097297297297	0.000379079365079365
40	90	10	0.000971563333333334	0.00137261
100	2	98	0.00039798	0.00477605551020408
1000	20	80	0.020302	0.041725737875

Σχήμα 36: Πίνακας Αποτελεσμάτων για την Ρυθμαπόδοση ανά Ταυτόχρονη Ανάγνωση και Εγγραφή

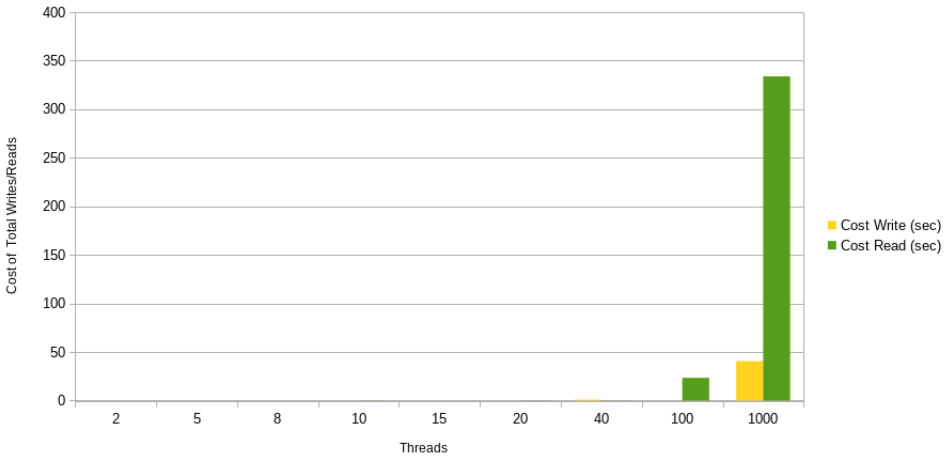


Σχήμα 37: Διάγραμμα Αποτελεσμάτων για την Ρυθμαπόδοση ανά Ταυτόχρονη Ανάγνωση και Εγγραφή

Παρατηρούμε ότι στο διάγραμμα 37 όταν η ρυθμαπόδοση είναι πολύ μικρή(απειροελάχιστη), το ορθογώνιο παραλληλόγραμμο δεν μπορεί να γίνει ορατό με γυμνό μάτι, με εξαίρεση την περίπτωση που χρησιμοποιούμε 2 νήματα, όπου η ρυθμαπόδοση ανά ανάγνωση είναι ίση με 0.

Threads	percentage write	percentage read	Cost Write (sec)	Cost Read (sec)
2	73	27	0.000104	0
5	50	50	0.000542	0.00044
8	60	40	0.016654	0.014032
10	30	70	0.037136	0.149492
15	42	58	0.009392	0.029996
20	37	63	0.028138	0.11941
40	90	10	0.874407	0.137261
100	2	98	0.039798	23.402672
1000	20	80	40.604	333.805903

Σχήμα 38: Πίνακας Αποτελεσμάτων Κόστους Συνολικών Ταυτόχρονων Αναγνώσεων και Εγγραφών



Σχήμα 39: Διάγραμμα Αποτελεσμάτων Κόστους Συνολικών Ταυτόχρονων Αναγνώσεων και Εγγραφών

Παρατηρούμε ότι στο διάγραμμα 39 όταν το συνολικό κόστος είναι πολύ μικρό(απειροελάχιστο), το ορθογώνιο παραλληλόγραμμο δεν μπορεί να γίνει ορατό με γυμνό μάτι, με εξαίρεση την περίπτωση που χρησιμοποιούμε 2 νήματα, όπου το κόστος των συνολικών αναγνώσεων είναι ίσο με 0.

6.4 Χρήσιμα συμπεράσματα

- Το συνολικό κόστος αναγνώσεων για οποιαδήποτε πλήθος αιτήσεων και οποιοδήποτε πλήθος νημάτων είναι μεγαλύτερο από το συνολικό κόστος εγγραφών, επειδή όταν αποκτά πρόσβαση στο memtable δεν βρίσκει το κλειδί και πρέπει να το αναζητήσει στο sst_table.

- Όσο αυξάνονται το πλήθος των αιτήσεων και το πλήθος των νημάτων, το μέσο κόστος ανάγνωσης/εγγραφής φθίνει, ενώ η ρυθμαπόδοση και το συνολικό κόστος αναγνώσεων/εγγραφών αυξάνονται. Αυτό όμως δεν ισχύει στην περίπτωση readwrite, γιατί τα αποτελέσματα εξαρτώνται άμεσα από το ποσοστό που δίνει ο χρήστης.

- Η ρυθμαπόδοση ανά εγγραφή/ανάγνωση είναι το αντίστροφο μέγεθος του μέσου κόστους εγγραφών/αναγνωσεων.

- Το μέσο κόστος αναγνώσεων για οποιαδήποτε πλήθος αιτήσεων και οποιοδήποτε πλήθος νημάτων είναι μικρότερο από το μέσο κόστος εγγραφών