Ντουμανόπουλος Χρήστος 2509
Μούρτος Ηλίας 2302

Λειτουργικά Συστήματα - Εργαστήριο 1

Στο συγκεκριμένο εργαστήριο υλοποιήσαμε έναν απλό πολυνηματικό διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής. Αρχικά μας δόθηκε ένας διακομιστής αποθήκευσης ζευγών κλειδιού-τιμής που λειτουργούσε σειριακά και εμείς έπρεπε να τον κάνουμε να λειτουργεί παράλληλα.

Τα κύρια αρχεία(αυτά που επηρέαζαν εμάς και αυτά που πειράξαμε εμείς εν τέλει) του διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής που μας δόθηκε ήταν ο server και ο client. Αρχικά ας περιγράψουμε πως ακριβώς λειτουργούν:

server: Ο server τρέχει όλη την ώρα περιμένοντας για νέες αιτήσεις. Αρχικά ανοίγει μια βάση δεδομένων, όπου θα αποθηκεύει τα κλειδιά με τις τιμές τους και θα τα ανακτά από εκεί όταν χρειάζεται. Στην συνέχεια μπαίνει σε μια επανάληψη και περιμένει κάποια νέα σύνδεση. Μόλις λάβει μια σύνδεση την “αποκρυπτογραφεί” δημιουργώντας μία αίτηση την οποία στην συνέχεια πρέπει να “φέρει εις πέρας”. Αυτές τις αιτήσεις τις λαμβάνει από τον client -για τον οποίο θα μιλήσουμε παρακάτω- μέσω κάποιων προκαθορισμένων sockets. Υπάρχουν 2 μορφών αιτήσεις,αιτήσεις PUT ή αιτήσεις GET. Σε μια αίτηση PUT ο client αποθηκεύει το ζεύγος κλειδιού-τιμής στην βάση δεδομένων την οποία δημιουργεί παραπάνω και επιστρέφει στον client "PUT OK\n", ενώ σε μια αίτηση GET διαβάζει από την βάση δεδομένων την τιμή που έχει το κλειδί της αίτησης και επιστρέφει στον client την τιμή του. Τέλος και αφού απαντήσει κλείνει την σύνδεση.

client:Ο client δέχεται κάποιες παραμέτρους που κρίνουν το αν θα στείλει μια ή πολλές αιτήσεις(πχ η παράμετρος -i είναι για να στείλει πολλές αιτήσεις,η παράμετρος -ο είναι για User Mode κοκ) και μετά τι αιτήσεις(-p για put -g για get) πρέπει να στείλει. Στην αρχή διαβάζει όλες τις παραμέτρους μια προς μια και θέτει τις τιμές αυτές στο πρόγραμμα ,στην συνέχεια ενημερώνει τις κατάλληλες παραμέτρους για την δημιουργία σύνδεσης και με βάση το αν πρέπει να στείλει μια αίτηση ή πολλές ετοιμάζει την αίτηση,στην μια με τα στοιχεία που του έχουν δοθεί (πχ PUT:station.125:5 για να αλλάξετε σε 5 την τιμή του κλειδιού station.125) ενώ στις πολλές (πχ ./client –a localhost –i k –p) στέλνει k φορές MAX_STATION_ID αιτήσεις όπου MAX_STATION_ID μια σταθερά που έχει οριστεί στην αρχή του προγράμματος και k ένας αριθμός που δίνεται από τον χρήστη.

Τρόπος λειτουργίας του παράλληλου προγράμματος:
Για να κάνουμε τον server να λειτουργεί παράλληλα, αρχικά θα δημιουργήσουμε ένα νήμα μέσα στην main, το οποίο θα παίρνει τις συνδέσεις όταν γίνονται και θα τις τοποθετεί στην ουρά μαζί με τον χρόνο που το πρόγραμμα τις έλαβε. Στην συνέχεια θα υπάρχουν τα νήματα που θα περιμένουν και μόλις κάποια αίτηση μπει στην ουρά θα την βγάζουν από εκεί και θα την εκτελούν μέσω του process_request που υπήρχε και πριν,ελαφρώς αλλαγμένο. Προφανώς για να γίνει αυτό προσθέσαμε κάποιες δομές και μεθόδους, πειράξαμε κάποιες από τις ήδη υπάρχουσες δομές και σε κάποια σημεία βάλαμε locks και conditions για την

ομαλή συνεργασία των νημάτων. Παρακάτω περιγράφεται σε κάθε μέθοδο ξεχωριστά ακριβώς τι κάνουμε.

Δομές που έχουμε προσθέσει στον server:

-queue: Καταρχήν έχουμε δημιουργήσει μια δομή queue που είναι η ουρά μας και εκεί θα αποθηκεύονται προσωρινά οι αιτήσεις. Η δομή αυτή αποτελείται από έναν πίνακα και από το rear για να ξέρουμε πόσα στοιχεία έχει μέσα.

-queueRequest: Μόλις λάβει μια νέα σύνδεση το νήμα που είναι υπεύθυνο να βάζει τις αιτήσεις στην ουρά για να τις πάρουν μετά τα άλλα νήματα και να τις διεκπεραιώσουν δημιουργεί μία queueRequest που αποτελείται από τον “αριθμό” της σύνδεσης και τον χρόνο που έγινε η σύνδεση ώστε να υπολογίσει μετά τον χρόνο αναμονής.

Μέθοδοι που έχουμε προσθέσει/πειράξει στον server:

-removeQ:(Το απλό remove σαν όνομα μας έβγαζε “Error : Conflicting types for 'remove'”, γι’αυτό βάλαμε και το Q) Αφαιρεί από την ουρά το 1ο αντικείμενο και πάει όλα τα υπόλοιπα μια θέση αριστερά ώστε να πάει το δεύτερο πρώτο το τρίτο δεύτερο κοκ. Αυτή η διαδικασία γίνεται μέσα σε lock γιατί πειράζει την ουρά που είναι κοινόχρηστη μεταξύ των νημάτων. Επίσης έχουμε προσθέσει μια μεταβλητή συνθήκης, ώστε όταν η ουρά είναι άδεια να περιμένει να μπει έστω και ένα στοιχείο πριν προχωρήσει.

-insert: Προσθέτει στο τέλος της ουράς αντικείμενα queueRequest. Όπως και στο removeQ η όλη διαδικασία γίνεται μέσα σε lock γιατί πειράζουμε την ουρά που είναι κοινόχρηστη και έχουμε προσθέσει μια μεταβλητή συνθήκης, ώστε όταν η ουρά είναι γεμάτη να περιμένει να βγει έστω και ένα στοιχείο πριν προχωρήσει.

-isQueueEmpty: Ελέγχει αν η ουρά είναι άδεια.

-isQueueFull:Ελέγχει αν η ουρά είναι γεμάτη.

-initializeThreads: Αρχικοποιεί τα νήματα,ο αριθμός των οποίων εξαρτάται από μια σταθερά που ορίζεται στην αρχή του προγράμματος καθώς επίσης και τις μεταβλητές του μέσου χρόνου αναμονής, καθώς και το μέσο χρόνο εξυπηρέτησης των αιτήσεων.

-threadsFunc: Είναι η μέθοδος που εκτελούν τα νήματα. Στην ουσία αυτό που έχει να κάνει κάθε νήμα είναι να πάρει μια αίτηση από την ουρά και να εκτελέσει την process_request για αυτήν την αίτηση. Επίσης κρατάει τους χρόνους για να υπολογίσει τον χρόνο αναμονής και τον χρόνο εκτέλεσης των αιτήσεων. Στο τέλος ενημερώνει τις κοινόχρηστες μεταβλητές του χρόνου αναμονής, του χρόνου εκτέλεσης καθώς και των ολοκληρωμένων αιτήσεων, χρησιμοποιώντας locks και εδώ γιατί οι μεταβλητές αυτές είναι κοινόχρηστες για όλα τα νήματα. Όλα αυτά είναι μέσα σε μία while ώστε κάθε νήμα αφού τελειώσει να περιμένει να ξαναμπεί κάποια νέα αίτηση στην ουρά.

-CtrlZ: Είναι η μέθοδος για τον τερματισμό του νήματος. Αρχικά υπολογίζει τον μέσο χρόνο αναμονής και τον μέσο χρόνο εξυπηρέτησης των αιτήσεων, τυπώνει το πλήθος των αιτήσεων που έχουν εξυπηρετηθεί καθώς και τους μέσους χρόνους αναμονής και εξυπηρέτησης αντίστοιχα και στην συνέχεια τερματίζει το πρόγραμμα.

-process_request: Στην process_request το πρόβλημα δημιουργούνταν στο σημείο που έγραφε και διάβαζε από την βάση δεδομένων. Εκεί μπορούμε να έχουμε πολλά νήματα να διαβάζουν αλλά ένα μόνο να γράφει και ποτέ ένα να γράφει και πολλά να διαβάζουν. Αυτό είναι το γνωστό πρόβλημα συγχρονισμού αναγνωστών-γραφέων, του οποίου ο αλγόριθμος είναι ο παρακάτω

lock-for-read operation in this setup is:[7][8][9]

- Input: mutex m, condition variable c, integer r (number of readers waiting), flag w (writer waiting).
- Lock m (blocking).
- While w:
  - wait c, m[a]
- Increment r.
- Unlock m.

The lock-for-write operation is similar, but slightly different (inputs are the same as for lock-for-read):[7][8][9]

- Lock m (blocking).
- While (w or $r > 0$):
  - wait c, m
- Set w to true.
- Unlock m.

Αυτό ακριβώς κάναμε και εκτός αυτού αφού κάθε νήμα τελειώσει με το διάβασμα/ανάγνωση από την βάση δεδομένων πάλι με ένα lock ενημερώνει(μειώνει κατά 1) τις μεταβλητές που μετράνε το πλήθος των γραφέων και των αναγνωστών αντίστοιχα.

-main: Εδώ στην ουσία πειράξαμε την ήδη υπάρχουσα while. Αφαιρέσαμε την process_request και το close της αίτησης,τα οποία πήγαν στην **threadsFunc.** Επίσης μέσα στην while όταν δεχτεί μια σύνδεση κρατάει τον χρόνο και δημιουργεί ένα queueRequest με τον αριθμό της αίτησης και τον χρόνο το οποίο βάζει στην ουρά. Δεν δημιουργήσαμε νέο νήμα με αυτήν την while σαν μέθοδο γιατί είναι ένα και δεν θα επηρέαζε κάπου την λειτουργικότητα του προγράμματος

Μέθοδοι που έχουμε προσθέσει/πειράξει στον client:

-threadsFunc: Ότι έκανε ο client πριν στα πολλαπλά get και put τώρα το κάνει εδώ μέσα. Η μόνη διαφορά είναι ότι αντί για count εδώ έχουμε μια μεταβλητή count1 η οποία όταν τροποποιείται υπάρχουν lock για τα άλλα νήματα γιατί είναι κοινόχρηστη. Ο λόγος που προσθέσαμε αυτήν την μεταβλητή είναι γιατί τα πολλαπλά get και put υλοποιούνται όταν καλούμε τον client με παράμετρο -i και μετά το i μπαίνει ένας αριθμός που είναι το πόσες φορές θα στείλει τις αιτήσεις. Αυτός ο αριθμός είναι ο αριθμός count. Μετά από αυτά το πρόγραμμα αρχικοποιεί στην main κάποιες παραμέτρους απαραίτητες για την σύνδεση που

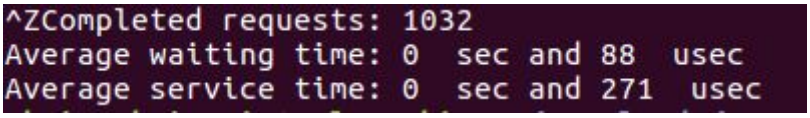
είναι κοινές για όλα τα νήματα και τρέχει αυτό το κομμάτι κώδικα κάθε νήμα ξεχωριστά. Στο σημείο που υπήρχε αυτός ο κώδικας βάζουμε count1=count και αρχικοποιούμε τα νήματα.

-main: Κάποιες μεταβλητές από την main που χρειάζονται για την δημιουργία σύνδεσης έγιναν global ώστε να είναι προσβάσιμες από όλα τα νήματα. Σε αυτές τις μεταβλητές δεν έχουμε lock γιατί απλά τις διαβάζουν τα νήματα αλλά δεν τις πειράζουν. Επίσης έγιναν οι αλλαγές που περιγράφηκαν παραπάνω και τέλος μετά το count1=count δημιουργούμε τα νήματα εκεί που θέλουμε να τα χρησιμοποιήσουμε και με την εντολή join περιμένουμε τα νήματα να τελειώσουν την εκτέλεσή τους πριν τερματίσουμε το πρόγραμμα.

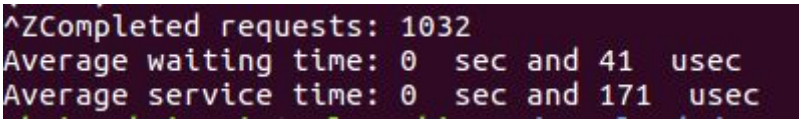
Έλεγχος που κάναμε:

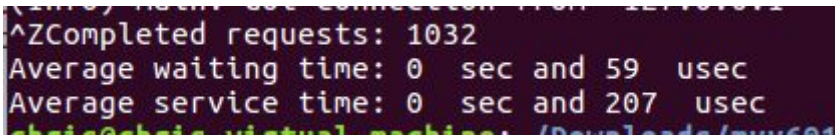
Αρχικά για να τεστάρουμε ότι όντως δουλεύει παράλληλα βάλαμε στον client 2 νήματα και του βάλαμε να στείλει 3 πακέτα αιτήσεων. Όπως αναμενόταν τα πρώτα 2 πηγαίναν σχετικά παράλληλα, ενώ το 3ο πήγε μετά μόνο του. Από εκεί και πέρα και για να ξεκαθαρίσουμε ποιες είναι κατάλληλες τιμές για τις μεταβλητές του server MAXSIZE(μέγιστο μέγεθος ουράς) και NUMTHREAD(αριθμός νημάτων) διατηρήσαμε σταθερό τον client στα 4 νήματα και στα 4 πακέτα αιτήσεων get και 4 πακέτα αιτήσεων put ώστε να έχουμε παντού σαν βάση το ίδιο για να παρατηρήσουμε τις διαφορές(κάναμε και κάποιες δοκιμές με 2 νήματα και 1,2,3 πακέτα αιτήσεων,με το να το ανεβάσουμε στα 4 αυξήσαμε τον χρόνο αναμονής σε μικρό αριθμό νημάτων στον server γιατί οι αιτήσεις ερχόταν πιο γρήγορα,αλλά καμία αίτηση δεν χανόταν).

MAXSIZE=10
NUMTHREAD=4

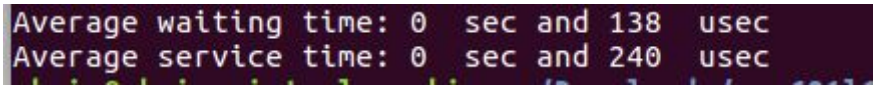


MAXSIZE=5
NUMTHREAD=4





MAXSIZE=15
NUMTHREAD=15



```
Average waiting time: 0  sec and 88  usec
Average service time: 0  sec and 284  usec
```

MAXSIZE=10
NUMTHREAD=12

```
^ZCompleted requests: 1032
Average waiting time: 0  sec and 88  usec
Average service time: 0  sec and 199  usec
```

MAXSIZE=4
NUMTHREAD=4

```
^ZCompleted requests: 1032
Average waiting time: 0  sec and 76  usec
Average service time: 0  sec and 228  usec
```

MAXSIZE=6
NUMTHREAD=4

```
^ZCompleted requests: 1032
Average waiting time: 0  sec and 61  usec
Average service time: 0  sec and 191  usec
```

```
^ZCompleted requests: 1032
Average waiting time: 0  sec and 50  usec
Average service time: 0  sec and 188  usec
```

Κάποιες μετρήσεις έγιναν 2 φορές για επιβεβαίωση των αποτελεσμάτων. Γενικά παρατηρούμε ότι οι διαφορές είναι πάρα πολύ μικρές αλλά όσο αυξάνουμε τον αριθμό των νημάτων και το μέγεθος της στοίβας τόσο αυξάνεται και ο μέσος χρόνος αναμονής. Ο μέσος χρόνος εξυπηρέτησης διαφέρει στα όρια του στατιστικού λάθους. Έτσι ιδανικά θέσαμε MAXSIZE=6 και NUMTHREAD=4 ως τις πιο κατάλληλες τιμές στο σύστημά μας, σύμφωνα με τις παραπάνω μετρήσεις.

Τα υπόλοιπα αρχεία δεν τα πειράξαμε καθόλου.