Τμήμα Μηχ. Η/Υ & Πληροφορικής **Εαρινό 2018**
Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

MYY601 Λειτουργικά Συστήματα

Ανακοίνωση: Δευτέρα 26 Φεβρουαρίου, Παράδοση: Δευτέρα 26 Μαρτίου στις 21:00

Εργαστήριο 1:Υλοποίηση ενός απλού πολύνηματικού διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής

1.1 Εισαγωγή

Στην παρούσα άσκηση, θα υλοποιήσετε έναν απλό πολυνηματικό διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής. Το λογισμικό σας θα μπορεί να δέχεται μια μικτή ακολουθία από ζεύγη κλειδιού τιμής για αποθήκευση στο διακομιστή και κλειδιά για ταυτόχρονη αναζήτηση. Επίσης, θα διατηρεί στατιστικά σχετικά με τους χρόνους εξυπηρέτησης των αιτήσεων. Ο κώδικας θα εκμεταλλεύεται τη βιβλιοθήκη νημάτων επιπέδου χρήστη POSIX threads του συστήματος Linux.

Αρχιτεκτονική Πελάτη-Διακομιστή

Θα χρησιμοποιήσετε προσέγγιση πελάτη-διακομιστή, στην οποία ο πελάτης προσδιορίζει το ζεύγος κλειδιού-τιμής που θα αποθηκευτεί ή το κλειδί τιμής που θα ανακτηθεί, και χρησιμοποιεί έναν υποδοχέα δικτύου (network socket) για να στείλει την αίτηση στο διακομιστή. Ο διακομιστής ανακτά από τη λαμβανόμενη αίτηση το ζεύγος κλειδιού-τιμής ή το καθορισμένο κλειδί, προσπελάζει τη βιβλιοθήκη αποθήκης για να αποθηκεύσει ή να ανακτήσει το ζεύγος, και επιστρέφει το αποτέλεσμα στον πελάτη. Μετά ο πελάτης εμφανίζει το αποτέλεσμα στην οθόνη. Η βασική λειτουργικότητα για τη σύνδεση πελάτη-διακομιστή δίνεται έτοιμη στο συνοδευτικό αρχείο myy601Lab1.zip.

Αποθήκη κλειδιού-τιμής

Η λύση σας βασίζεται στην αποθήκη κλειδιού-τιμής ανοιχτού κώδικα KISSDB. Το API της αποθήκης περιλαμβάνει τις ακόλουθες τέσσερις κλήσεις: `KISSDB_open()`, `KISSDB_close()`, `KISSDB_put()` και `KISSDB_get()`.

- Η κλήση `KISSDB_open()` δημιουργεί μια νέα αποθήκη σύμφωνα με τις καθορισμένες παραμέτρους, ή την ανοίγει αν υπάρχει ήδη μία στο καθορισμένο μονοπάτι.
- Η κλήση `KISSDB_close()` κλείνει την αποθήκη.
- Η `KISSDB_put()` εισάγει ένα ζεύγος κλειδιού-τιμής με αντικατάσταση, αν το κλειδί υπάρχει ήδη στην αποθήκη.
- Η `KISSDB_get()` κάνει αναζήτηση του καθορισμένου κλειδιού, και αν βρεθεί επιστρέφει την ανακτημένη τιμή στην παράμετρο ενδιάμεσης μνήμης της κλήσης.

Ο πλήρης κώδικας της βιβλιοθήκης αποθήκης κλειδιού-τιμής KISSDB περιλαμβάνεται στο παρεχόμενο δείγμα κώδικα, στα αρχεία `kissdb.[ch]` με συνοδευτικές επεξηγήσεις στα αρχεία `SPEC.txt` και `README.md`. Είναι σημαντικό να τονίσουμε ότι ο κώδικας KISSDB υποτίθεται ότι διασυνδέεται απευθείας σε μια εφαρμογή αντί να δουλεύει ως ανεξάρτητος δικτυωμένος διακομιστής.

Αναζήτηση

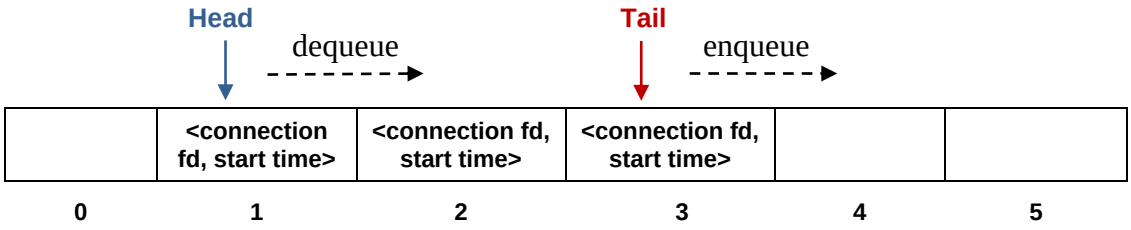
Η βασική εργασία της άσκησης είναι να οικοδομήσετε τον πολυνηματικό διακομιστή αποθήκευσης κλειδιού-τιμής. Η απαιτούμενη προσπέλαση της αποθήκης κλειδιού-τιμής είναι πολύ απλή. Ο κώδικας διακομιστή ανοίγει την αποθήκη και εφαρμόζει την αιτούμενη λειτουργία PUT ή GET με το καθορισμένο ζεύγος κλειδιού-τιμής ή κλειδί, αντίστοιχα. Υποθέτουμε ότι ο διακομιστής θα πρέπει να είναι ικανός να υποστηρίξει τις

λειτουργίες PUT ακολουθιακά, τη μία μετά την άλλη. Όμως, θα πρέπει να είναι ικανός να υποστηρίξει πολλαπλές λειτουργίες GET ταυτόχρονα τη μία σε σχέση με τις άλλες.

Πολυνηματική υλοποίηση

Ο διακομιστής σας θα έχει δομή παραγωγού-καταναλωτή. Ένα νήμα παραγωγού περιμένει για εισερχόμενες αιτήσεις σύνδεσης, ενώ ένα προκαθορισμένο πλήθος από νήματα καταναλωτή είναι υπεύθυνα για την εξυπηρέτηση των αιτήσεων αναζήτησης. Όταν ο διακομιστής λάβει μία νέα σύνδεση, το νήμα παραγωγού προσθέτει σε μια κοινόχρηστη ουρά μία περιγραφή της σύνδεσης. Στη συνέχεια, ένα νήμα καταναλωτή απομακρύνει την περιγραφή αυτή από την ουρά, ανακτά το κλειδί και κάνει την αναζήτηση κλειδιού χρησιμοποιώντας τη βιβλιοθήκη αποθήκης.

Αναλυτικότερα, κάθε νήμα καταναλωτή παραμένει αδρανές μέχρι να φτάσει μια νέα αίτηση στο διακομιστή. Όταν φτάνει η αίτηση, το νήμα παραγωγού προετοιμάζει μια δομή ως περιγραφέα αίτησης που περιέχει τον *περιγραφέα αρχείου* της εισερχόμενης σύνδεσης καθώς και τον *χρόνο έναρξής* της. Ο περιγραφέας έπειτα εισάγεται σε *κοινόχρηστη ουρά (First-In-First-Out)* με προκαθορισμένο μέγιστο μέγεθος **QUEUE_SIZE**. Μια ουρά σταθερού μέγιστου μεγέθους μπορεί να υλοποιηθεί με έναν πίνακα, όπως φαίνεται στο παρακάτω σχήμα:



Στη συνέχεια, τα νήματα καταναλωτή ειδοποιούνται ότι μια καινούρια αίτηση είναι διαθέσιμη για εξυπηρέτηση. Το νήμα που θα κάνει την εξυπηρέτηση αφαιρεί τον περιγραφέα από την ουρά. Έπειτα, ανακτά το κλειδί από την αίτηση και εκτελεί τη διαδικασία της αναζήτησης. Κάθε φορά που ολοκληρώνεται μια αναζήτηση, το νήμα αποθηκεύει σε μια θέση μνήμης προσωρινής αποθήκευσης (buffer) το αποτέλεσμα της αναζήτησης.

Επιπλέον, όταν ολοκληρωθεί η αναζήτηση το νήμα υπολογίζει το *χρόνο αναμονής* της αίτησης (ο χρόνος που η αίτηση παρέμεινε στην ουρά μέχρι να ξεκινήσει η αναζήτηση), καθώς και το *χρόνο εξυπηρέτησης* (ο χρόνος που απαιτήθηκε για την αναζήτηση της λέξης σε όλα τα αρχεία). Στη συνέχεια, οι χρόνοι αυτοί προστίθενται στις αντίστοιχες *κοινόχρηστες* μεταβλητές **total_waiting_time** και **total_service_time**, ενώ προσαυξάνεται και η *κοινόχρηστη* μεταβλητή **completed_requests** που περιέχει το πλήθος των αιτήσεων που έχουν εξυπηρετηθεί.

Τέλος, το νήμα στέλνει τη συμβολοσειρά με το αποτέλεσμα της αναζήτησης στον πελάτη.

Διαχείριση ουράς

Η ουρά αιτήσεων είναι προσβάσιμη σε όλα τα νήματα του διακομιστή. Προκειμένου να διατηρηθεί η ουρά σε συνεπή κατάσταση, μόνο ένα νήμα επιτρέπεται να την προσπελάζει κάθε φορά. Για να το διασφαλίσετε αυτό, όλες οι τροποποιήσεις και προσπελάσεις που κάνετε στην κοινόχρηστη δομή θα πρέπει να προστατεύονται με χρήση των συναρτήσεων **pthread_mutex_lock()** και **pthread_mutex_unlock()**. Με τον ίδιο τρόπο θα πρέπει να προστατευτούν και όλα τα υπόλοιπα *κοινόχρηστα* δεδομένα.

Είναι πιθανό η ουρά αιτήσεων να αδειάσει. Τότε, τα νήματα καταναλωτή θα περιμένουν καλώντας **pthread_cond_wait()** μέχρι να φτάσει μια νέα αίτηση. Ο παραγωγός ειδοποιεί τα νήματα καταναλωτή καλώντας **pthread_cond_signal()**. Για τη λειτουργία αυτή θα χρειαστείτε μια μεταβλητή συνθήκης, π.χ. **non_empty_queue**. Με παρόμοιο τρόπο, όταν η ουρά γεμίζει, ο παραγωγός περιμένει καλώντας **pthread_cond_wait()** μέχρι τουλάχιστο μία κενή θέση να γίνει διαθέσιμη στην ουρά. Τότε, το νήμα καταναλωτή που εξυπηρετεί την αντίστοιχη αίτηση ειδοποιεί τον παραγωγό με την κλήση της συνάρτησης **pthread_cond_signal().** Για το σκοπό αυτό, θα δημιουργήσετε μια αντίστοιχη μεταβλητή συνθήκης, π.χ. **non_full_queue**.

Έλεγχος Ταυτοχρονισμού

Προκειμένου να διατηρήσετε συνεπή την αποθήκη κλειδιού-τιμής, είναι αναγκαίο να υλοποιήσετε μια απλή λύση συγχρονισμού αναγνωστών-γραφέων χρησιμοποιώντας pthreads. Η λύση σας θα πρέπει να μετράει με τη μεταβλητή **reader_count** τους αναγνώστες (απεριόριστοι) μέσα στην αποθήκη κλειδιού-τιμής (ως κρίσιμης περιοχής) και με τη μεταβλητή **writer_count** τους γραφείς (το πολύ 1) που τροποποιούν την αποθήκη κλειδιού-τιμής. Χρησιμοποιώντας μεταβλητές συνθήκης, ο συγχρονισμός θα πρέπει να επιτρέπει μόνο ένα γραφέα τη φορά να τροποποιεί την αποθήκη κλειδιού-τιμής, και πολλαπλούς αναγνώστες να τη διαβάζουν, αλλά όχι γραφείς και αναγνώστες την ίδια στιγμή στην αποθήκη κλειδιού-τιμής. Η λύση σας δεν απαιτείται να παρέχει κάποια προτεραιότητα είτε στους αναγνώστες ή στους γραφείς, άρα οποιαδήποτε λύση από αυτή την άποψη είναι αποδεκτή.

Σήματα

Το σύστημα UNIX ορίζει ένα προκαθορισμένο σύνολο από σήματα που μπορούν να σταλούν από μία διεργασία σε μία άλλη, με αποτελέσμα η δεύτερη διεργασία να διακοπεί και να λάβει το σήμα εκτελώντας τον κώδικα που αντιστοιχεί στο συγκεκριμένο σήμα. Η αποστολή ενός σήματος σε μία *διεργασία* γίνεται με τη συνάρτηση **kill()** (`man 2 kill`). Η διεργασία που θα λάβει το σήμα μπορεί να το διαχειριστεί με τον προκαθορισμένο τρόπο, να το αγνοήσει, ή να εκτελέσει κώδικα που όρισε ο χρήστης. Η συνάρτηση **signal()** καλείται για να προσδιορίσει τον αριθμό του σήματος και τον τρόπο με τον οποίο θα διαχειριστούμε το συγκεκριμένο σήμα. Για παράδειγμα, μία διεργασία μπορεί να αγνοήσει το σήμα **SIGINT** (Control+C) εκτελώντας την κλήση συστήματος:

```
signal(SIGINT, SIG_IGN);
```

Η διεργασία μπορεί επίσης να χειριστεί το σήμα **SIGINT** με το δικό της κώδικα, ορίζοντας μια συνάρτηση ως δεύτερη παράμετρο στην **signal()**.

Χρονομέτρηση

Μία διεργασία μπορεί να λάβει την τρέχουσα ώρα μέσω της συνάρτησης **gettimeofday()** (`man 2 gettimeofday`). Η συνάρτηση αυτή επιστρέφει το πλήθος των δευτερολέπτων και των μικροδευτερολέπτων που έχουν περάσει από την αρχή της ώρας του UNIX (1 Ιανουαρίου 1970, 12 πμ). Για παράδειγμα:

```
#include <sys/time.h>
#include <stdio.h>

int main(void) {
        struct timeval tv;
        gettimeofday(&tv, NULL);
        printf("Seconds and microseconds since 1/1/1970 12a.m.: %ld sec, %ld
usec.\n", tv.tv_sec, tv.tv_usec);
```

}

όπου,

```
struct timeval {
        long tv_sec;   /* seconds */
        long tv_usec;/* microseconds */
}
```

Η συνάρτηση αυτή μπορεί να χρησιμοποιηθεί για τη *χρονομέτρηση* μιας λειτουργίας. Αυτό επιτυγχάνεται υπολογίζοντας τη διαφορά μεταξύ των χρόνων που επιστρέφει η **gettimeofday()** πριν την έναρξη της λειτουργίας και μετά την ολοκλήρωσή της.

1.2 Προετοιμασία

Φρεσκάρετε τις γνώσεις σας στην πρότυπη βιβλιοθήκη εισόδου/εξόδου της γλώσσας C. Βεβαιωθείτε ότι κατανοείτε τις συναρτήσεις της βιβλιοθήκης Pthreads που απαιτούνται για τη δημιουργία και συγχρονισμό των νημάτων. Κατεβάστε το αρχείο myy601Lab1.zip, όπου θα βρείτε βασικό κώδικα πελάτη-διακομιστή και τη συλλογή των αρχείων κειμένου. Αποσυμπιέστε το αρχείο .zip και μεταγλωττίστε τον πελάτη και το διακομιστή με:

> make all

Ξεκινήστε το διακομιστή με:

> ./server &

Στη συνέχεια, ξεκινήστε τον πελάτη με

> ./client –a localhost –i 1 –p
για να αποθηκεύσετε μερικά δεδομένα (π.χ., σταθμών αισθητήρων) και μετά εκτελέστε

> ./client –a localhost –i 1 –g
για να ανακτήσετε όλα τα αποθηκευμένα δεδομένα, ή

> ./client –a localhost –o GET:station.125
για να ανακτήσετε την αποθηκευμένη τιμή για το κλειδί `station.125`, και

> ./client –a localhost –o PUT:station.125:5
για να αλλάξετε σε 5 την τιμή του κλειδιού `station.125`.

Τόσο ο πελάτης όσο και ο διακομιστής με τη βιβλιοθήκη αποθήκης κλειδιού-τιμής είναι πλήρως λειτουργικά στο παρεχόμενο δείγμα κώδικα. Η εργασία σας είναι να υλοποιήσετε τον ταυτοχρονισμό του διακομιστή για το χειρισμό των αιτήσεων PUT και GET από πολλαπλούς πελάτες.

1.3 Άσκηση

Για την άσκηση θα πρέπει να δημιουργήσετε έναν απλό πολυνηματικό διακομιστή αποθήκης κλειδιού-τιμής. Προκειμένου να διατηρήσετε την πολυπλοκότητα της άσκησης ελεγχόμενη, μπορείτε να προχωρήσετε σταδιακά:

i. Πειραματιστείτε με την έκδοση μονής διεργασίας του διακομιστή αποθήκης που σας δίνεται. Μια μοναδική διεργασία είναι υπεύθυνη να δεχτεί αρχικά μια μικτή ακολουθία από ζεύγη κλειδιού-τιμής για αποθήκευση και κλειδιά για αναζήτηση. Ο πελάτης τυπώνει το αποτέλεσμα της κάθε λειτουργίας.
ii. Θεωρήστε την περίπτωση που έχετε ένα νήμα παραγωγού που δέχεται τις αιτήσεις σύνδεσης και ένα νήμα καταναλωτή που εκτελεί την αιτούμενη λειτουργία. Υποθέστε ότι το νήμα παραγωγού κάθε φορά που φτάνει μια νέα αίτηση σύνδεσης εμφανίζει το *χρόνο έναρξης* της σύνδεσης, τυπώνοντας την τρέχουσα ώρα, και έπειτα δημιουργεί ένα νέο νήμα καταναλωτή για την εξυπηρέτηση της αίτησης. Το νήμα

καταναλωτή δέχεται ως όρισμα τον περιγραφέα αρχείου της εισερχόμενης δικτυακής σύνδεσης. Έπειτα, το νήμα καταναλωτή εκτελεί την αιτούμενη λειτουργία, και τυπώνει το *χρόνο εξυπηρέτησης* της αίτησης. Τέλος, επιστρέφει στον πελάτη το αποτέλεσμα της λειτουργίας και *τερματίζει*.

iii. Υποθέστε ότι δημιουργούνται πολλαπλά νήματα καταναλωτή εκ των προτέρων. Ορίστε μια *δομή* κόμβου ουράς που θα περιέχει τον *περιγραφέα αρχείου* της σύνδεσης πελάτη καθώς και το *χρόνο έναρξης* της σύνδεσης. Χρησιμοποιήστε μια ουρά τέτοιων δομών για την επικοινωνία μεταξύ παραγωγού και καταναλωτών. Προσδιορίστε τον κώδικα αμοιβαίου αποκλεισμού ώστε η ουρά να προσπελάζεται από ένα νήμα κάθε φορά. Προσθέστε κώδικα συγχρονισμού για τις συνθήκες άδειας και γεμάτης ουράς. Παρόμοια υλοποιήστε την λύση αναγνωστών-γραφέων που επιτρέπει είτε σε ένα γραφέα να τροποποιήσει την αποθήκη κλειδιού-τιμής, ή πολλαπλούς ταυτόχρονους αναγνώστες, αλλά όχι γραφείς και αναγνώστες την ίδια στιγμή.

Τροποποιήστε κατάλληλα τον παραγωγό ώστε κατά την λήψη μίας νέας σύνδεσης να ενημερώνει το *χρόνο έναρξής* της στη δομή σύμφωνα με την τρέχουσα ώρα. Επίσης, προσθέστε κώδικα βάσει του οποίου κάθε νήμα μετά την ολοκλήρωση της εξυπηρέτησης μίας αίτησης υπολογίζει το χρόνο αναμονής και εξυπηρέτησης της αίτησης και ενημερώνει τις *κοινόχρηστες* μεταβλητές **total_waiting_time, total_service_time** και **completed_requests**, τις οποίες θα πρέπει να προσθέσετε στον κώδικά σας. Διασφαλίστε ότι ο κώδικας χειρίζεται σωστά ταυτόχρονες εισαγωγές και αναζητήσεις. *Επαναχρησιμοποιήστε* τα νήματα για διαφορετικές εισαγωγές και αναζητήσεις λέξεων.

Για απλότητα *κατά την εκσφαλμάτωση* μπορείτε να τρέξετε τις λειτουργίες PUT και GET σε διαφορετικές φάσεις της λειτουργίας του διακομιστή. Για παράδειγμα, κατά την αρχική φάση φόρτωσης, ο διακομιστής υποστηρίζει μόνο λειτουργίες PUT και, κατά τη διάρκεια της φάσης ανάγνωσης, ο διακομιστής υποστηρίζει μόνο ταυτόχρονες λειτουργίες GET. Τελικά όμως θα πρέπει να υποστηρίζετε ταυτόχρονη εκτέλεση των PUT και GET.

iv. Προσθέστε κώδικα βάσει του οποίου όταν ληφθεί το σήμα **SIGTSTP** (Control+Z) ο διακομιστής υπολογίζει και εκτυπώνει το *πλήθος* των αιτήσεων που έχουν εξυπηρετηθεί, το *μέσο χρόνο αναμονής*, καθώς και το *μέσο χρόνο εξυπηρέτησης* των αιτήσεων. Στη συνέχεια το πρόγραμμα τερματίζει.

v. Για να ελέγξετε επίσης την ορθότητα του κώδικα σας, τροποποιήστε κατάλληλα τον πελάτη ώστε να στέλνει πολλαπλές αιτήσεις ταυτόχρονα. Αυτό μπορεί να γίνει είτε δημιουργώντας πολλαπλά νήματα ή πολλαπλές διεργασίες, κάθε μία από τις οποίες στέλνει μία διαφορετική αίτηση στον διακομιστή.

1.4 Τι θα παραδώσετε

Μπορείτε να προετοιμάσετε τη λύση ατομικά ή σε ομάδες των δύο. Ακόμη και αν η ομάδα έχει δύο μέλη θα υποβάλλετε μόνο από τον λογαριασμό του ενός. Υποβολή μετά την προθεσμία μειώνει το βαθμό 10% κάθε ημέρα μέχρι 50%. Για παράδειγμα, εάν υποβάλλετε 1 ώρα μετά την προθεσμία ο μέγιστος βαθμός σας γίνεται 9 στους 10. Αν υποβάλλετε μια εβδομάδα μετά την προθεσμία, ο μέγιστος βαθμός πέφτει στο 5 από 10. Υποβάλλετε τη λύση σας με την εντολή

/usr/local/bin/turnin lab1_18@myy601 README.pdf file1 …

Το αρχείο κειμένου **README.pdf** περιγράφει τη λειτουργία του πολυνηματικού διακομιστή αποθήκευσης, τις βασικές δομές και συναρτήσεις που χρησιμοποιήσατε και τα αρχεία πηγαίου κώδικα στα οποία περιέχονται. Επιπλέον περιγράφει τις κανονικές και ειδικές περιπτώσεις που χρησιμοποιήσατε για να εκσφαλματώσετε τον κώδικα.

Συμπεριλάβετε στο αρχείο README.pdf τα ονόματα των μελών της ομάδας που υποβάλλουν τη λύση.

Το μέγεθος της ουράς και το πλήθος των νημάτων είναι παράμετροι επηρεάζουν τη λειτουργία του συστήματος σας. Για τον κώδικα που θα παραδώσετε αρκεί να θέσετε τις παραμέτρους αυτές σε κάποιες τιμές που θεωρείται λογικές. Ωστόσο, θα πρέπει να δοκιμάσετε κάποιο εύρος τιμών και να αναφέρετε τις παρατηρήσεις σας (π.χ. πλήθος αιτήσεων που εξυπηρετούνται, μέσος χρόνος αναμονής κτλ) στο README.pdf που θα παραδώσετε.

Υποβάλετε τα αρχεία όλου του πηγαίου κώδικα που απαιτείται για να μεταγλωττιστεί ο πελάτης και ο διακομιστής. Μην υποβάλετε **.o** ή εκτελέσιμα αρχεία, αλλά συμπεριλάβετε αρχεία scripts που ελέγχουν την ορθότητα του κώδικα. Ο κώδικας πρέπει να μεταγλωττίζεται με **gcc** και να τρέχει σε μηχανές Linux του Τμήματος.