Τμήμα Μηχ. Η/Υ & Πληροφορικής **Εαρινό 2020**
Πανεπιστήμιο Ιωαννίνων **Διδάσκων: Σ. Αναστασιάδης**

MYY601 Λειτουργικά Συστήματα

Ανακοίνωση: Τρίτη 3 Μαρτίου, Παράδοση: Παρασκευή 20 Μαρτίου στις 21:00
Εργαστήριο 1:Υλοποίηση ενός απλού πολυνηματικού διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής

1. Εισαγωγή

Στην παρούσα άσκηση, θα υλοποιήσετε έναν απλό πολυνηματικό διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής. Το λογισμικό σας θα δέχεται μια μικτή ακολουθία αιτήσεων αποθήκευσης ζευγών κλειδιού-τιμής και αναζήτησης κλειδιού. Επίσης, θα διατηρεί στατιστικά σχετικά με τους χρόνους εξυπηρέτησης των αιτήσεων. Ο κώδικας θα βασίζεται στην βιβλιοθήκη POSIX threads του Linux.

1.1 Αποθήκη κλειδιού-τιμής

Θα χρησιμοποιήσετε αρχιτεκτονική 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 στον 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 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 ενδιάμεση μνήμη 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. Ο κώδικας KISSDB είναι σχεδιασμένος να διασυνδέεται απευθείας σε μια εφαρμογή αντί να δουλεύει ως ανεξάρτητος διακομιστής.

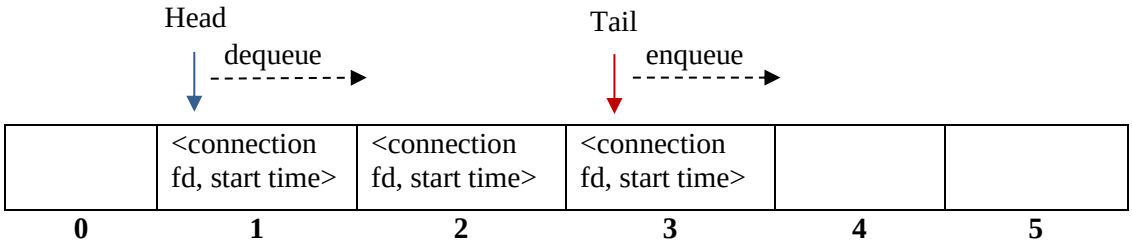
Η βασική εργασία της άσκησης είναι να υλοποιήσετε τον *πολυνηματικό* διακομιστή αποθήκευσης κλειδιού-τιμής. Η απαιτούμενη προσπέλαση της αποθήκης κλειδιού-τιμής είναι πολύ απλή. Ο κώδικας διακομιστή ανοίγει την αποθήκη και εφαρμόζει την αιτούμενη λειτουργία PUT ή GET με το καθορισμένο ζεύγος κλειδιού-τιμής ή κλειδί, αντίστοιχα. Ο διακομιστής θα πρέπει να υποστηρίζει τις λειτουργίες αποθήκευσης PUT ακολουθιακά την μία μετά την άλλη, ενώ θα υποστηρίζει πολλαπλές λειτουργίες αναζήτησης GET ταυτόχρονα μεταξύ τους.

1.2 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. Όταν ο διακομιστής λάβει μία νέα σύνδεση, το νήμα παραγωγού προσθέτει σε μια κοινόχρηστη ουρά έναν περιγραφέα της σύνδεσης. Στη συνέχεια, ένα νήμα καταναλωτή απομακρύνει τον περιγραφέα από την ουρά, και εξάγει το κλειδί ή το ζεύγος κλειδιού-τιμής προκειμένου να εκτελέσει την λειτουργία GET ή PUT στην βιβλιοθήκη KISSDB.

Αναλυτικότερα, τα νήματα καταναλωτή παραμένουν αδρανή 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 ή το ζεύγος κλειδιού-τιμής από την αίτηση και εκτελεί την καθορισμένη λειτουργία GET ή PUT. Επιπλέον, όταν ολοκληρωθεί μια λειτουργία το νήμα υπολογίζει το *χρόνο αναμονής* της αίτησης (τον χρόνο που η αίτηση παρέμεινε στην ουρά μέχρι να ξεκινήσει η αναζήτηση), καθώς και το *χρόνο εξυπηρέτησης* (τον χρόνο που απαιτήθηκε για την εκτέλεσης της λειτουργίας στην βιβλιοθήκη KISSDB). 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 λειτουργίας στον πελάτη.

1.3 Διαχείριση ουράς και Έλεγχος Ταυτοχρονισμού

Η ουρά αιτήσεων είναι προσβάσιμη σε όλα τα νήματα του διακομιστή. Προκειμένου να διατηρηθεί η ουρά σε συνεπή κατάσταση, θα πρέπει να επιτρέπεται μόνο σε έναν παραγωγό την φορά να εισάγει αιτήσεις και έναν καταναλωτή την φορά να αφαιρεί αιτήσεις. *Δηλαδή η λύση σας θα επιτρέπει ένα παραγωγό και έναν καταναλωτή να τροποποιούν ταυτόχρονα την ουρά, αλλά θα εμποδίζει περισσότερους από έναν παραγωγό ή περισσότερους από έναν καταναλωτή να κάνουν τροποποιήσεις ταυτόχρονα.* Για να το διασφαλίσετε αυτό, όλες οι τροποποιήσεις που γίνονται από τους παραγωγούς και τους καταναλωτές θα πρέπει να προστατεύονται με χρήση των συναρτήσεων *pthread_mutex_lock()* και *pthread_mutex_unlock()* που θα παίρνουν την κατάλληλη μεταβλητή αμοιβαίου αποκλεισμού ως παράμετρο.

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*.

Προκειμένου να διατηρήσετε συνεπή την αποθήκη κλειδιού-τιμής, είναι αναγκαίο να υλοποιήσετε μια απλή λύση συγχρονισμού αναγνωστών-γραφέων χρησιμοποιώντας μεταβλητές συνθήκης των pthreads. Η λύση σας θα πρέπει να μετράει με τη μεταβλητή *reader_count* τους αναγνώστες (απεριόριστοι) που τρέχουν ταυτόχρονα 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.

1.4 Σήματα και Χρονομέτρηση

Το σύστημα UNIX ορίζει ένα προκαθορισμένο σύνολο από σήματα που μπορούν να σταλούν από μία διεργασία σε μία άλλη, με αποτέλεσμα 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. 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).

2. Προετοιμασία

Κατεβάστε την εικονική μηχανή MYY601Lab1VM.zip (1GB) και αποσυμπιέστε την στον τοπικό σας δίσκο (ή σε μνήμη USB με τουλάχιστο 5GB διαθέσιμο χώρο). Η εικονική μηχανή έχει εγκατεστημένο την διανομή 10 του Debian 64-bit Linux. Χρησιμοποιήστε VMware Player v15 για να τρέξτε την εικονική μηχανή. Η μηχανή είναι ρυθμισμένη να χρησιμοποιήσει 1 πυρήνα και 1GB RAM, αλλά αυτές τις ρυθμίσεις μπορείτε να τις αλλάξετε μέσα από το μενού Settings της εικονικής μηχανής. Για να εκμεταλλευτείτε το γραφικό παραθυρικό περιβάλλον της μηχανής, εξασφαλίστε ότι έχετε βάλει την εικονική μηχανή σε κατάσταση πλήρους οθόνης πιέζοντας το αντίστοιχο εικονίδιο πάνω αριστερά. Μπορείτε να εισέλθετε ως απλός χρήστης με το όνομα myy601 (και το ίδιο ως συνθηματικό). Σε περίπτωση που χρειαστεί να εγκαταστήσετε επιπλέον πακέτα στο σύστημα Linux, μπορείτε να εισέλθετε ως χρήστης root (με συνθηματικό myy601) και να τρέξετε στην γραμμή εντολής apt install <package>. Η εικονική μηχανή είναι προετοιμασμένη με εγκατεστημένη την γλώσσα C και τον φυλλομετρητή ιστού Firefox. Για διευκόλυνσή σας, μπορείτε να κάνετε αντιγραφή-επικόλληση (copy-paste) κείμενο ή αρχεία μεταξύ του λειτουργικού συστήματα επισκέπτη που τρέχει στην εικονική μηχανή (Debian Linux) και του λειτουργικού συστήματος φιλοξενίας, για παράδειγμα, προκειμένου να πάρετε τα αρχεία που αλλάξετε και να τα υποβάλετε (turnin) για την εξέταση.

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. Στον κατάλογο ~myy601/myy601l1 της εικονικής μηχανής θα βρείτε τον βασικό κώδικα πελάτη-διακομιστή. 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 προερχόμενων από πολλαπλούς πελάτες.

3. Υλοποίηση

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 αρχική έκδοση διακομιστή μονής διεργασίας που σας δίνεται. Μια μοναδική διεργασία δέχεται μια μικτή ακολουθία αποτελούμενη από αιτήσεις αποθήκευσης

ζευγών κλειδιού-τιμής και αιτήσεις αναζήτησης κλειδιών. Ο πελάτης τυπώνει το αποτέλεσμα της κάθε λειτουργίας.

ii. Υλοποιήστε τον διακομιστή με δύο νήματα, 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. Υλοποιήστε τον διακομιστή με πολλαπλά νήματα καταναλωτή που δημιουργούνται 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.
Σχεδιάστε και υλοποιήστε την απαιτούμενη λογική αμοιβαίου αποκλεισμού προκειμένου η ουρά να προσπελάζεται από <u>το πολύ έναν παραγωγό</u> και <u>το πολύ έναν καταναλωτή</u> κάθε φορά. Προσθέστε κώδικα συγχρονισμού για τις συνθήκες άδειας και γεμάτης ουράς. *Παρόμοια υλοποιήστε την λύση αναγνωστών-γραφέων που επιτρέπει σε ένα γραφέα να τροποποιεί την αποθήκη κλειδιού-τιμής, ή πολλαπλούς αναγνώστες να κάνουν ταυτόχρονες αναζητήσεις, αλλά όχι γραφείς και αναγνώστες να εκτελούνται την ίδια στιγμή.*
Τροποποιήστε κατάλληλα τον παραγωγό ώστε κατά την λήψη μίας νέας σύνδεσης να ενημερώνει το *τρέχοντα χρόνο έναρξης* στη δομή. 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 κώδικα. Διασφαλίστε ότι ο κώδικας χειρίζεται σωστά ταυτόχρονες εισαγωγές και αναζητήσεις. *<u>Επαναχρησιμοποιήστε τα νήματα για διαφορετικές εισαγωγές και αναζητήσεις ζευγών κλειδιού-τιμής.</u>*

iv. Προσθέστε κώδικα βάσει του οποίου όταν αποσταλεί στον διακομιστή 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 δημιουργώντας πολλαπλά νήματα καθένα από τα οποία στέλνει μία διαφορετική ακολουθία αιτήσεων στον διακομιστή.*

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_20@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 με την υλοποίησή σας. 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 που επηρεάζουν τη λειτουργία του διακομιστή 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). *Επιπλέον στο README.pdf συμπεριλάβετε γραφικές παραστάσεις που να δείχνουν πώς επηρεάζεται η απόδοση όταν αλλάζετε κάποια παράμετρο εισόδου.*

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 στην εικονική μηχανή που σας δίνεται ή σε μηχανές Linux του Τμήματος.