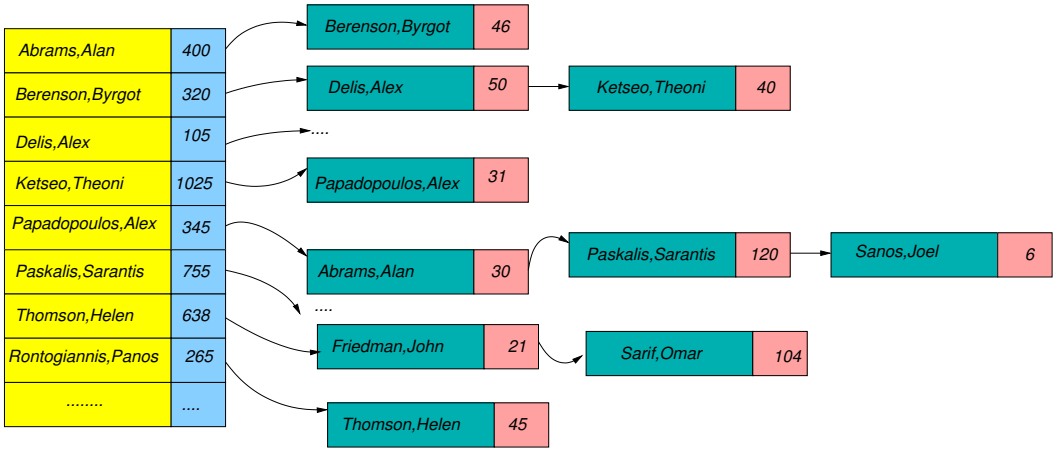
Κ24: Προγραμματισμός Συστήματος
3η Εργασία – Εαρινό Εξάμηνο 2016
Προθεσμία Υποβολής: Κυριακή 5 Ιουνίου 2016 Ώρα 23:59

Εισαγωγή στην Εργασία:

Ο στόχος της εργασίας αυτής είναι να εξοικειωθείτε με την χρήση sockets και νημάτων (threads) στην δημιουργία μιας διαδικτυακής εφαρμογής. Επιπλέον στα πλαίσια της εργασίας θα πρέπει να συγχρονίσετε τα νήματα που η εφαρμογή σας θα χρησιμοποιήσει.

Καλείστε να υλοποιήσετε ένα πρόγραμμα server (bankserver) που θα κάνει εξομοίωση μιας λειτουργίας διαχείρισης τράπεζας. Το πρόγραμμα θα διατηρεί πληροφορία για ονόματα και τους λογαριασμούς τους στην κυρίως μνήμη. Κάθε λογαριασμός διατηρεί το *τρέχον υπόλοιπό* του και μια λίστα από τα ποσά (συνολικά, ανά λογαριασμό) που έχει λάβει από άλλους λογαριασμούς.



Σχήμα 1: Πληροφορίες που διατηρούνται στον bankserver

Ο πίνακας στα αριστερά του Σχήματος 1 είναι απλούστευση. Θα χρειαστείτε δομές για γρήγορη πρόσβαση σε τυχαίο λογαριασμό (π.χ. ένα hash table).

Ο bankserver θα δέχεται εντολές από προγράμματα – πελάτες (bankclients) μέσω συνδέσεων sockets. Τα προγράμματα–πελάτες θα δίνουν εντολές για δημιουργία λογαριασμού με αρχικό υπόλοιπο, μεταφορά χρημάτων μεταξύ λογαριασμών και εκτύπωση πληροφοριών σχετικών με τους λογαριασμούς. Θα υπάρχει μία σύνδεση (socket) ανά πελάτη, αλλά πάνω από αυτή θα στέλνονται πιθανά πολλαπλές εντολές.

Το πρόγραμμα bankserver αναλαμβάνει:

- Τη δημιουργία της αρχικής δομής που θα αποθηκεύονται οι πληροφορίες λογαριασμού και μεταφοράς ποσών.
- Τη λήψη εντολών από πελάτες (bankclients) που λειτουργούν ταυτόχρονα για τη διαχείριση των λογαριασμών.
- Την εκτέλεση των εντολών και την επιστροφή σε κάθε bankclient του όποιου αποτελέσματος (επιτυχία, αποτυχία, αδυναμία εκτέλεσης).

Το πρόγραμμα bankclient αναλαμβάνει:

- Την αποστολή εντολής δημιουργίας νέου λογαριασμού με νέο όνομα, αρχικό υπόλοιπο.
- Την αποστολή εντολής μεταφοράς ποσού μεταξύ λογαριασμών

- Την αποστολή εντολής λήψης πληροφορίας για το υπόλοιπο λογαριασμού/λογαριασμών.
- Την εμφάνιση των αποτελεσμάτων στο `stdout`.

Περιγραφή του bankserver:

Ο bankserver κατά την εκκίνηση του, θα περιμένει για εξωτερικές συνδέσεις σε μια θύρα (`port`) που θα καθορίζεται από τη γραμμή εντολών. Κάθε νέα σύνδεση που δέχεται ο bankserver μέσω δικτυακής επικοινωνίας (π.χ. κάθε νέο socket που επιστρέφει η `accept` για επικοινωνία με τον αντίστοιχο πελάτη) θα τοποθετείται σε μια ουρά εκτέλεσης. Η τοποθέτηση αυτή μπορεί να γίνεται από το βασικό νήμα του bankserver που ονομάζουμε master/administrative thread.

Θεωρούμε ότι ο bankserver διαθέτει ένα thread pool από "workers" σταθερού μεγέθους. Το κάθε τέτοιο worker thread του pool αναλαμβάνει μία σύνδεση/πελάτη που προέρχεται από την ουρά εκτέλεσης. Κάθε τέτοιο worker thread είναι υπεύθυνο να εκτελέσει τις εντολές που λαμβάνει και να στείλει τις κατάλληλες απαντήσεις στον αντίστοιχο πελάτη, μέσω του socket. Όταν ένας πελάτης ολοκληρώσει τη δουλειά του, το αντίστοιχο worker thread που διεκπεραίωσε τις εντολές, ξαναγίνεται διαθέσιμο στο επίπεδο του bankserver. Στο σημείο αυτό, το εν λόγω νήμα επιστρέφει στο worker thread pool και περιμένει για να αναλάβει νέα σύνδεση από την ουρά εκτέλεσης όταν μια τέτοια σύνδεση/πελάτης υπάρχει ή εμφανιστεί.

Επίσης θεωρούμε ότι η ουρά εκτέλεσης έχει ένα μέγιστο μέγεθος. Σε περίπτωση που η ουρά εκτέλεσης είναι κενή, τα διαθέσιμα worker threads θα πρέπει να περιμένουν. Σε περίπτωση που η ουρά εκτέλεσης έχει δεχθεί τουλάχιστον μια σύνδεση -κάτι που ο master/administrative thread θα πρέπει να γνωρίζει- ένας απο τους (διαθέσιμους) worker threads μπορεί να αναλάβει εργασία. Στο μάλλον απίθανο σενάριο υπό φυσιολογικές συνθήκες η ουρά να βρεθεί γεμάτη, ο πελάτης απλά θα μπλοκάρει/περιμένει μέχρι να εξυπηρετηθεί.

Στο σχεδιασμό σας μπορείτε να υιοθετήσετε ένα η και πιο πολλούς master ή management threads.

Περιγραφή του bankclient:

Ο bankclient συνδέεται με τον bankserver μέσω διεύθυνσης `IP` και θύρας που καθορίζονται από τη γραμμή εντολών. Στη συνέχεια ξεκινάει να δέχεται εντολές είτε μέσω `stdin` είτε μέσω αρχείου εντολών και εκτυπώνει τις απαντήσεις που λαμβάνει από τον bankserver στο `stdout`.

Λεπτομερείς Προδιαγραφές για Γραμμές Εντολής:

Το πρόγραμμα bankserver εκτελείται από τη γραμμή εντολών ως εξής:
`$ ./bankserver -p <port> -s <thread_pool_size> -q <queue_size>` όπου:

- `<port>`: Η θύρα στην οποία θα ακούει το socket για εισερχόμενες συνδέσεις από πελάτες για εισαγωγή εντολών.
- `<thread_pool_size>`: Ο αριθμός των worker threads στο thread pool.
- `<queue_size>`: Το μέγεθος της ουράς εκτέλεσης

Το πρόγραμμα bankclient εκτελείται από τη γραμμή εντολών ως εξής:
`$ ./bankclient -h <server_host> -p <server_port> -i <command_file>` όπου:

- `<server_host>`: Η συμβολική διεύθυνση (όνομα) του μηχανήματος του server.
- `<server_port>`: Η θύρα στην οποία ακούει ο server για εξωτερικές συνδέσεις.
- `<command_file>`: Το αρχείο που περιέχει εντολές που θα αποσταλούν προς τον server.

Τα ορίσματα της γραμμής εντολών είναι υποχρεωτικά και για τα δύο εκτελέσιμα προγράμματα. Επιπλέον, τα ζευγάρια ορισμάτων μπορεί να δίνονται σε διαφορετική σειρά από αυτή της εκφώνησης.

Μηνύματα που ανταλλάσσονται:

Ο bankclient δέχεται τις εξής εντολές:

1. `add_account <init_amount> <name> [delay]`
   Προσθέτει ένα λογαριασμό με αρχικό υπόλοιπο.
2. `add_transfer <amount> <src_name> <dst_name> [delay]`
   Προσθέτει μεταφορά ποσού μεταξύ δύο λογαριασμών.
3. `add_multi_transfer <amount> <src_name> <dst_name1> <dst_name2> ... [delay]`
   Προσθέτει μεταφορά ποσού από ένα λογαριασμό σε πολλούς.
4. `print_balance <name>`
   Επιστρέφει το υπόλοιπο του λογαριασμού.
5. `print_multi_balance <name1> <name2> ...`
   Επιστρέφει τα υπόλοιπα πολλών λογαριασμών.
6. `sleep <time>`
   Καθυστερεί τη μετάδοση της επόμενης εντολής στον bankserver
7. `exit` Τερματίζει το πρόγραμμα–πελάτη bankclient.

όπου:

- <name>: Το χαρακτηριστικό όνομα λογαριασμού.
- <init_amount>: Το αρχικό υπόλοιπο του λογαριασμού.
- <delay>: Η ελάχιστη διάρκεια της επεξεργασίας της εντολής από τον server σε msec.
- <src_name>: Ο λογαριασμός από τον οποίο πραγματοποιείται η μεταφορά.
- <dst_name>: Ο λογαριασμός στον οποίο πραγματοποιείται η μεταφορά.
- <sleep>: Ο χρόνος σε msec που θα πρέπει να περιμένει ο bankclient πριν στείλει την επόμενη εντολή στον bankserver.

Η παράμετρος `delay` στις πρώτες 3 εντολές είναι προαιρετική. Όλες οι άλλες παράμετροι είναι υποχρεωτικές. Το νόημα της παραμέτρου `delay` είναι ότι εξομοιώνεται καθυστέρηση ενός πραγματικού συστήματος. Δηλαδή θα πρέπει ο εξομοιωτής σας (δηλαδή ο bankserver) να συμπεριφέρεται σαν πραγματικά η εντολή να παίρνει `delay` msec για να εκτελεστεί. Συγκεκριμένα αυτό σημαίνει ότι αν κλειδώνετε κάποια δομή για ταυτόχρονη προσπέλαση από πολλά νήματα, θα πρέπει να υπάρχει αντίστοιχη καθυστέρηση ανάμεσα στο κλείδωμα και το ξεκλείδωμα.

Η 6η `sleep` και η 7η εντολή `exit` εκτελούνται τοπικά στον bankclient. Όλες οι άλλες εντολές μεταδίδονται στον bankserver.

Ο bankserver μπορεί να απαντήσει με τα εξής μηνύματα:

1. Για την εντολή `add_account`:
   - Σε περίπτωση επιτυχίας:
     `Success. Account creation (name:init_amount)`
   - Σε περίπτωση αποτυχίας:
     `Error. Account creation failed (name:init_amount)`
2. Για την εντολή `add_transfer`:
   - Σε περίπτωση επιτυχίας:
     `Success. Transfer addition (src_name:dst_name:amount[:delay])`
   - Σε περίπτωση αποτυχίας:
     `Error. Transfer addition failed (src_name:dst_name:amount[:delay])`
3. Για την εντολή `add_multi_transfer`:

- Σε περίπτωση επιτυχίας:
  `Success. Multi-Transfer addition (src_name:amount[:delay])`
- Σε περίπτωση αποτυχίας:
  `Error. Multi-Transfer addition failed (src_name:amount[:delay])`

4. Για την εντολή `print_balance`:
   - Σε περίπτωση επιτυχίας:
     `Success. Balance (name:amount)`
   - Σε περίπτωση αποτυχίας:
     `Error. Balance (name)`
5. Για την εντολή `print_multi_balance`:
   - Σε περίπτωση επιτυχίας:
     `Success. Multi-Balance (name1/amount1:name2/amount2:...)`
   - Σε περίπτωση αποτυχίας:
     `Error. Multi-Balance (name1:name2:...)`
6. Για οποιαδήποτε άλλη είσοδο/εντολή:
   `Error. Unknown command`

Σημαντικές Οδηγίες:

Η βασική πρόκληση της εργασίας είναι να υποστηρίξετε γρήγορη πρόσβαση στους λογαριασμούς (δηλ. σε τυχαίο λογαριασμό) και παραλληλισμό μεταξύ των νημάτων. Δηλαδή η ιδεώδης λύση δεν θα περιέχει μία και μόνο κλειδαριά για ολόκληρη τη δομή που κρατάει τους λογαριασμούς, αλλά αρκετές μικρότερες και ανεξάρτητες κλειδαριές. Για να υποστηρίξετε κάτι τέτοιο, αν μια λειτουργία επηρεάζει πολλούς λογαριασμούς, θα πρέπει να προστατευθούν όλοι από ταυτόχρονη πρόσβαση πολλών νημάτων.

Η κάθε εντολή πρέπει να συμπεριφέρεται σαν *αδιαίρετη μονάδα*. Για παράδειγμα αν υπάρχει σε εξέλιξη μια εντολή του τύπου `add_multi_transfer` και πριν τελειώσει η εν λόγω εντολή φτάσει μια εντολή `print_multi_balance`, δεν πρέπει η δεύτερη να φαίνεται σαν να έγινε στη μέση της πρώτης (με κάποιους λογαριασμούς ανανεωμένους και κάποιους όχι). Φυσικά η λύση σας θα πρέπει να εξασφαλίζει ότι το πρόγραμμα δεν έχει race conditions, δηλαδή δεν βλέπει ασυνεπή δεδομένα (π.χ. ατελείς δομές) σε χαμηλό επίπεδο και δεν έχει deadlock.

Διαδικαστικά:

- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux της σχολής.
- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο `piazza.com` για να δείτε ερωτήσεις/απαντήσεις/διευκρινήσεις που δίνονται σχετικά με την άσκηση. Η παρακολούθηση του φόρουμ στο `piazza.com` είναι *υποχρεωτική*.

Τι πρέπει να Παραδοθεί:

1. Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα `source files, header files` και `Makefile`.
   Προσοχή: φροντίστε να τροποποιήσετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με την εντολή:
   `$ chmod 644 OnomaEponymoProject3.tar`
   Για πιο πολλές λεπτομέρειες δείτε τη σελίδα του μαθήματος.
2. Ένα `README` (απλό text αρχείο) με κάποια παραδείγματα μεταγλώττισης και εκτέλεσης του προγράμματός σας. Επίσης, μπορείτε να συμπεριλάβετε άλλες διευκρινίσεις που κρίνετε απαραίτητες (για τυχόν παραδοχές που έχετε κάνει, κτλ.).

3. Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στο παραπάνω `README`.

Τι θα Βαθμολογηθεί:

1. Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης.
2. Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα.
3. Η χρήση `Makefile` και η κομματιαστή σύμβολο-μετάφραση (separate compilation).

Άλλες Σημαντικές Παρατηρήσεις:

1. Οι εργασίες είναι *ατομικές*.
2. Όποιος υποβάλλει/δείχνει κώδικα που δεν έχει γραφτεί από την ίδια/τον ίδιο *μηδενίζεται* στο μάθημα.
3. Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, *αντιγραφή κώδικα* (οποιασδήποτε μορφής) είναι κάτι που *δεν επιτρέπεται* και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα *απλά παίρνει μηδέν* στο μάθημα. Αυτό ισχύει για *όλους όσους εμπλέκονται*, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
4. Το πρόγραμμά σας θα πρέπει να γραφτεί σε C ή C++.