Εργασία 1 – Request-reply με σημασιολογία at most once και πολλούς εξυπηρετητές

Αναπτύξτε λογισμικό για την υποστήριξη του σχήματος αλληλεπίδρασης request-reply μέσα από το παρακάτω ενδεικτικό API (βλέπε διαλέξεις). Η υλοποίησή σας πρέπει να σχεδιαστεί ως ξεχωριστή υπηρεσία ή ξεχωριστό τμήμα λογισμικού που δεν είναι στενά συνδεδεμένο με την εφαρμογή (η υλοποίηση πρέπει να χρησιμοποιεί τα δικά της νήματα έτσι ώστε να εκτελείται ανεξάρτητα από την εφαρμογή).

Διασύνδεση πελάτη	`int RequestReply (int svcid, void *reqbuf, int reqlen, void *rspbuf, int *rsplen);`
Διασύνδεση εξυπηρετητή	`int register (int svcid);`
	`int unregister (int svcid);`
	`int getRequest (int svcid, void *buf, int *len);`
	`void sendReply (int reqid, void *buf, int len);`

Το πρωτόκολλο που θα σχεδιάσετε πρέπει να είναι ανεξάρτητο γλώσσας προγραμματισμού / πλατφόρμας εκτέλεσης. Η ανακάλυψη του εξυπηρετητή από τον πελάτη πρέπει να γίνει με UDP/IP multicast, και η μεταφορά δεδομένων ανάμεσα στις δύο πλευρές πρέπει να γίνει πάνω από UDP/IP. Μπορείτε να υποθέσετε ότι κάθε αίτηση/απάντηση χωράει σε ένα UDP datagram (δεν χρειάζεται fragmentation και re-assembly). Δεν μπορείτε να υποθέσετε κάποιο άνω όριο για τον χρόνο επεξεργασίας των αιτήσεων στον εξυπηρετητή. Η παρεχόμενη σημασιολογία πρέπει να είναι at most once ακόμα και σε περίπτωση επανεκκίνησης του εξυπηρετητή μετά από βλάβη. Αν η πλευρά του πελάτη, μετά από προσπάθειες, δεν λάβει δείγμα ζωής από τον εξυπηρετητή, υποθέτει ότι παρουσίασε βλάβη και εγκαταλείπει. Αντίστοιχα, η πλευρά του εξυπηρετητή πρέπει να εντοπίζει και να χειρίζεται πιθανές βλάβες του πελάτη.

Δοκιμάστε την υλοποίησή σας μέσω μιας απλής εφαρμογής για τον έλεγχο πρώτων αριθμών (primality test), όπου ο χρόνος επεξεργασίας μπορεί να διαφέρει σημαντικά αναλόγως με τον αριθμό που ελέγχεται κάθε φορά. Ο κώδικας της εφαρμογής πρέπει να αναπτυχθεί ξεχωριστά από την υλοποίηση του παραπάνω API. Ο εξυπηρετητής στο επίπεδο της εφαρμογής πρέπει να είναι πολυνηματικός με άνω όριο 3 worker threads.

Κάντε πειράματα για: (α) να μετρήσετε την καθυστέρηση μιας «μηδενικής» αλληλεπίδρασης όπου τα μηνύματα σε επίπεδο εφαρμογής είναι κενά και η επεξεργασία μιας αίτησης σε επίπεδο εφαρμογής είναι μηδενική, και (β) να επιβεβαιώσετε τη δυνατότητα ταυτόχρονης εξυπηρέτησης πολλών αιτήσεων / πελάτων από τον ίδιο εξυπηρετητή.

Επέκταση: Επεκτείνετε την υλοποίησή σας για να υποστηρίζει πολλούς εξυπηρετητές που προσφέρουν την ίδια υπηρεσία, και έτσι ώστε να γίνεται ομοιόμορφη κατανομή φορτίου. Αυτό μπορεί να επιτευχθεί με ανταλλαγή πληροφορίας απ' ευθείας ανάμεσα στους εξυπηρετητές ή μέσω μιας ξεχωριστής οντότητας που μεσολαβεί ανάμεσα στους πελάτες και τους εξυπηρετητές (για την επιλογή του εξυπηρετητή που θα στείλει την αίτηση ο πελάτης). Πρέπει να υποστηρίζεται η δυναμική προσθήκη/αφαίρεση εξυπηρετητών.

Ιδανικά, δεν πρέπει να γίνουν αλλαγές στο παραπάνω API και η δοκιμαστική εφαρμογή που ήδη φτιάξατε για να δοκιμάσετε τη βασική υλοποίηση πρέπει να μπορεί να μεταφράζεται/εκτελείται χωρίς καμία αλλαγή.

Κάντε πειράματα για: (α) να μετρήσετε την επιπλέον καθυστέρηση που πιθανώς προκαλείται για μια «μηδενική» αλληλεπίδραση, και (β) να επιβεβαιώσετε τη δυνατότητα κατανομής των αιτήσεων στους διαθέσιμους εξυπηρετητές

Υλοποίηση: Η πλευρά του API πελάτη πρέπει να υλοποιηθεί σε διαφορετική γλώσσα από αυτήν του εξυπηρετητή. Μια από τις δύο πλευρές (όποια θέλετε) πρέπει να υλοποιηθεί σε C. Η άλλη πλευρά μπορεί να υλοποιηθεί σε όποια γλώσσα προγραμματισμού επιθυμείτε, αρκεί να έχει ρητή υποστήριξη για νήματα. Ακολουθήστε τις οδηγίες παράδοσης εργασιών (υπάρχουν στο eclass).

Φροντιστήριο/συζήτηση: **Πέμπτη 3 Μαρτίου 2022**
Ημερομηνία παράδοσης: **Σάββατο 19 Μαρτίου 2022, 22:00**