Εργασία 1 – Ασύγχρονο request-reply με σημασιολογία at most once και ελαστική εξυπηρέτηση

Αναπτύξτε μεσοστρωματικό λογισμικό για την υποστήριξη του ασύγχρονου σχήματος επικοινωνίας request-reply, ως ξεχωριστή βιβλιοθήκη που υποστηρίζει μια κατάλληλη διασύνδεση προγραμματισμού, π.χ.:

Διασύνδεση πελάτη	`int sendRequest (int svcid, void *buf, int len);`
	`int getReply (int reqid, void *buf, int *len, int block);`
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);`

Η ανακάλυψη της πλευράς του εξυπηρετητή από την πλευρά του πελάτη πρέπει να γίνει με UDP multicast, και η μετάδοση αιτήσεων/απαντήσεων ανάμεσα σε πελάτη και εξυπηρετητή πρέπει να γίνει πάνω από UDP. Τα αντίστοιχα πρωτόκολλα θα τα σχεδιάσετε εσείς. Δεν μπορείτε να κάνετε υποθέσεις για το περιεχόμενο των αιτήσεων/απαντήσεων της εφαρμογής, αλλά μπορείτε να υποθέσετε ότι κάθε αίτηση/απάντηση χωράει σε ένα UDP datagram. Δεν μπορείτε να κάνετε υποθέσεις για τον χρόνο επεξεργασίας των αιτήσεων στον εξυπηρετητή.

Η παρεχόμενη σημασιολογία πρέπει να είναι at most once. Αν μετά από έναν αριθμό προσπαθειών η πλευρά του εξυπηρετητή δεν στείλει την αναμενόμενη επιβεβαίωση/απάντηση, η πλευρά του πελάτη απλά υποθέτει ότι ο εξυπηρετητής παρουσίασε βλάβη. Αντίστοιχα, η πλευρά του εξυπηρετητή πρέπει να εντοπίζει και να χειρίζεται κατάλληλα τυχόν βλάβες του πελάτη.

Δοκιμάστε την υλοποίηση σας μέσω μιας εφαρμογής για τον έλεγχο του αν ένας αριθμός είναι πρώτος (primality test), όπου εύκολα μπορείτε να γίνουν δοκιμές για διαφορετικούς χρόνους επεξεργασίας (αναλόγως με τον αριθμό που πρέπει να ελεγχθεί). Ο εξυπηρετητής της εφαρμογής μπορεί να υλοποιηθεί με μονονηματικό ή πολυνηματικό τρόπο. Κάντε δοκιμές στην αρχή με έναν και μετά με περισσότερους πελάτες καθένας από τους οποίους σε πρώτη φάση στέλνει πολλές αιτήσεις και στην συνέχεια περιμένει να λάβει τα αποτελέσματα με την σειρά που αυτά τυχαίνει να επιστρέφονται από τον εξυπηρετητή.

Στην συνέχεια, επεκτείνετε την υλοποίηση του μεσοστρωματικού λογισμικού έτσι ώστε οι εξυπηρετητές μιας υπηρεσίας να ενεργοποιούνται / απενεργοποιούνται δυναμικά, ανάλογα με τον φόρτο του συστήματος (αριθμό αιτήσεων που έχουν παραληφθεί στο επίπεδο του ενδιάμεσου λογισμικού αλλά δεν βρίσκονται σε επεξεργασία από την εφαρμογή). Αυτό μπορεί να επιτευχθεί είτε με απ' ευθείας ανταλλαγή πληροφορίας ανάμεσα στους εξυπηρετητές, είτε μέσω μιας «ρυθμιστικής αρχής» που μεσολαβεί ανάμεσα στους πελάτες και τους εξυπηρετητές. Ιδανικά, η νέα λειτουργικότητα δεν θα πρέπει να επιφέρει αλλαγές στην διασύνδεση προγραμματισμού του μεσοστρωματικού λογισμικού, και ο κώδικας της παραπάνω εφαρμογής που ήδη γράψατε δεν θα χρειάζεται αλλαγές για να τρέξει με την νέα έκδοση του ενδιάμεσου λογισμικού.

Σκεφτείτε κατάλληλα σενάρια δοκιμής της επιπλέον λειτουργικότητας, με την εφαρμογή που ήδη φτιάξατε. Π.χ., στην αρχή δημιουργήστε έναν αριθμό εξυπηρετητών (καθένας από τους οποίους έχει άνω όριο νημάτων αν είναι πολυνηματικός). Στην συνέχεια δημιουργήστε έναν πελάτη που στέλνει αιτήσεις με δυναμικά αυξομειούμενο ρυθμό – εναλλακτικά μπορείτε να έχετε έναν δυναμικά αυξομειούμενο αριθμό πελατών καθένας από τους οποίους στέλνει αιτήσεις με σταθερό ρυθμό. Καταγράψετε το πως ενεργοποιούνται και απενεργοποιούνται οι εξυπηρετητές, καθώς το πως μεταβάλλεται η μέση καθυστέρηση επεξεργασίας των αιτήσεων στο χρόνο, ανάλογα με τον φόρτο του συστήματος.

Μπορείτε να χρησιμοποιήσετε όποια γλώσσα προγραμματισμού επιθυμείτε. Ο κώδικας σας θα πρέπει να μεταφράζεται/εκτελείται κανονικά στο περιβάλλον Linux του εργαστηρίου. Εκτός από τον κώδικα σας, πρέπει να παραδώσετε μια συνοπτική παρουσίαση της υλοποίησης και των μετρήσεων.

Φροντιστήριο/συζήτηση: **Πέμπτη 21 Φεβρουαρίου 2019**
Ημερομηνία παράδοσης: **Σάββατο 9 Μαρτίου 2019, 22:00**