Κατανεμημένα Συστήματα

Άσκηση 2

Μέθοδος αιτιακά διατεταγμένης πολυεκπομπής

(causally ordered multicasting)

Γιώργος Μύταρος

ΑΜ: 2312

Email: gmitaros@gmail.com, cs122312@cse.uoi.gr

Στην άσκηση αυτή υλοποιούμε την μέθοδο της αιτιακά διατεταγμένης πολυεκπομπής. Η λογική της υλοποίησης είναι η εξής. Έχουμε 3 διεργασίες(0, 1, 2) και όλες ξεκινάνε με το διανυσματικό ρολόι [0, 0, 0]. Έπειτα συνδέονται μεταξύ τους οι 3 διεργασίες και η διεργασία 0 ξεκινάει να πολυεκπεμπει 20 ερωτήσεις ανά 3''. Στη συνέχεια για κάθε ερώτηση μια διεργασία από τις 1 και 2 πολυεκπέμπουν μια απάντηση εναλλάξ. Τέλος η κάθε διεργασία θα γράφει σε ένα ένα αρχείο τα μηνύματα που τις παραδίδει το λογισμικό αιτιακά διατεταγμένης εκπομπής.

Αποστολή μηνύματος

Το κάθε μήνυμα στέλνεται με την sendUpdate(self, message) όπου εκεί στέλνω το μήνυμα, που παίρνει σαν όρισμα η συνάρτηση, σε κάθε συνδεδεμένο κόμβο.

Λήψη μηνύματος

Τα μηνύματα λαμβάνονται από την **dataReceived(self, data)** η οποία λαμβάνει το μήνυμα το επεξεργάζεται κατάλληλα και το προωθεί στην **causallOrder(self, msg)** όπου και γίνεται η υλοποίηση της αιτιακά διατεταγμένης πολυεκπομπής.

def causallOrder(self, msg)

Εδώ γίνεται η υλοποίηση της αιτιακά διατεταγμένης πολυεκπομπής. Όταν λάβει ένα μήνυμα η **dataReceived** το προωθεί εδώ. Γενικά τα μηνύματα που στέλνονται είναι 2 τύπου, ερώτηση και απάντηση. Όταν λάβει ένα μήνυμα η **casuallOrder** αυτή αρχικά ελέγχει αν είναι τύπου «**Order**» (δηλ. απάντηση). Αν είναι απάντηση λοιπόν τότε το προωθεί στην λίστα **orderQueue** η οποία υλοποιεί μια Priority Queue. Αλλιώς αν δεν είναι απάντηση και είναι ερώτηση αυτό το μήνυμα τότε περνάμε σε ένα HashMap με όνομα **msgWaitingQueue** την εξής πληροφορία. Σαν key έχω το id του μηνύματος και σαν value έχω το περιεχόμενο του μηνύματος. Έπειτα αν είμαι η **διεργασία 1 (κανονικά αυτό θα έπρεπε να γινόταν εναλλάξ με την 2 αλλά δε το πρόσεξα και το έκανα να απαντάει μόνο η 1)** ελέγχω αρχικά αν αυτό το μήνυμα που έλαβα είναι σύμφωνα με τα ρολόγια το επόμενο που θα έπρεπε να λάβω. Εάν είναι true ο έλεγχος που περιέγραψα τότε κάνω multicast ένα μήνυμα τύπου Order (δηλ. μια

απάντηση) και αλλάζω και το ρολόι μου με το μεγαλύτερο ανάμεσα στην τιμή του ρολογιού μου και στην τιμή του ρολογιού του μηνύματος. Αλλιώς το κρατάω σε μια λίστα **globalEvent** για να το επεξεργαστώ αργότερα.

Στην συνέχεια και αφού είμαι ακόμα μέσα στην συνθήκη ότι **είμαι η διεργασία 1** για κάθε μηνυμα μέσα στην globalEvent ελέγχω αν αυτό το μήνυμα είναι σύμφωνα με τα ρολόγια το επόμενο που θα έπρεπε να λάβω και αν ισχύει αυτό τότε κάνω multicast ένα μήνυμα τύπου Order (δηλ. μια απάντηση), αλλάζω και το ρολόι μου με το μεγαλύτερο ανάμεσα στην τιμή του ρολογιού μου και στην τιμή του ρολογιού του μηνύματος και μετά το βγάζω από την λίστα **globalEvent** .

Στην περίπτωση που δεν είμαι η (διεργασία 1) τότε αλλάζω και τις 3 τιμές του διανυσματικού ρολογιού μου με το μεγαλύτερο ανάμεσα στην τιμή του ρολογιού μου και στην τιμή του ρολογιού του μηνύματος.

def loop(self)
Αυτή η συνάρτηση δημιουργεί ανά 2” για 20 φορές ένα μήνυμα και μέσω της **sendUpdate** το κάνει multicast μόνο η διεργασία 0. Επίσης μετά από κάθε αποστολή καλεί την **deliverMessage** η οποία ελέγχει εάν υπάρχει κάποιο μήνυμα για εξαγωγή και εάν υπάρχει το εξάγει.

def deliverMessage(self)
Αυτή η συνάρτηση εφόσον η **orderQueue** έχει μέσα στοιχεία ελέγχει εάν το μήνυμα που έχει στην **orderQueue** (δηλ έχει λάβει απάντηση για αυτό) υπάρχει και στην **msgWaitingQueue** (δηλ έχει λάβει το μήνυμα) και ελέγχει επίσης εάν το id του μηνύματος ταιριάζει με τα μηνύματα που έχει λάβει η διεργασία. Τότε μόνο εξάγει το μήνυμα και αυξάνει και το groupSequenceNumber δηλαδή πόσα μηνύματα έχει λάβει.