Κατανεμημένα Συστήματα
Άσκηση #1

Υλοποίηση της ολικά διατεταγμένης πολυεκπομπής
(totally ordered multicasting)

Γιώργος Μύταρος
AM: 2312
email: gmitaros@gmail.com, cs122312@cse.uoi.gr

Στην άσκηση αυτη υλοποιούμε την μέθοδο της ολικά διατεταγμένης πολυεκπομπής. Η λογική της υλοποιησης ειναι η εξής. Αρχικά ξεκιναω την καθε διεργασία με τιμή ρολογιού ίση με το id της, η 0 δηλαδη θα εχει clock = 0 , η 1 δηλαδη θα εχει clock = 1 και η 2 δηλαδη θα εχει clock = 2. Έπειτα βάζω την κάθε μια να στέλνει εναν αριθμό ανα 2" και οταν λαμβάνει ενα μήνυμα τον αριθμο δηλαδη που στέλνω να τον καταχωρει στην τοπική μεταβλητή κάθε διεργασίας **counter** οπου την αρχικοποιώ σε ολες τις διεργασίες με 0.

Αποστολή μηνύματος

Το κάθε μηνυμα στέλνεται με την **sendUpdate(self, message)** οπου εκει στέλνω το μηνυμα που παιρνει σαν ορισμα η συναρτηση σε καθε συνδεδεμενο κόμβο. Επίσης το μηνυμα αυτο το στέλνω ατυπα και στον εαυτό της για να μπορεσω να το κανω αργοτερα ευκολότερα pop απο την ουρά και να γινει τελικα deliver το μηνυμα στην διεργασία.

Λήψη μηνύματος

Τα μηνύματα λαμβάνονται απο την **dataReceived(self, data)** η οποια λαμβάνει το μήνυμα το επεξεργάζεται κατάλληλα και το προωθεί στην **totalOrder(self, msg)** οπου και γίνεται η υλοποιηση της ολικά διατεταγμένης πολυεκπομπής.

def totalOrder(self, msg)
Εδώ γινεται η υλοποιηση της ολικά διατεταγμένης πολυεκπομπής. Οταν λαβει ενα μηνυμα η dataReceived το προωθει εδω. Αρχικά το ρολοι της διεργασίας αλλάζει μέσα απο αυτη την σχέση *self.clock = max(int(msg.clock), int(self.clock)) + 1* που σημαίνει ότι θα πάρει την μεγιστη τιμή αναμεσα στο ρολοι που εχει τωρα και στο ρολοι του μηνυματος και μετα θα αυξηθει κατα 1.
Έπειτα δημιουργούμε ενα **id** με 2 τιμές(το id του δημιουργου και το ρολόι του δημιουργου του μηνύματος). Στη συνέχεια βλεπω αν αυτο το id υπαρχει μεσα στον πινακα acks και αν δεν υπαρχει το εισαγω μαζι με το μηνυμα, ενώ αν υπαρχει ηδη αυτο το id και το μηνυμα ειναι ack τοτε το προσθετω στον *self.acks[id]* ενά αν δεν ειναι ack το βάζω στην αρχη του *self.acks[id].*
Μετά με μια αλλη **if** ελέγχω εαν το μηνυμα **ΔΕΝ** ειναι ack και τοτε το κανω push στην queue μου.

Σε μία αλλη if ελέγχω εαν εχω λάβει 3 acks. 2 απο της αλλες διεργασες και 1 απο οταν το στελνει ατυπα στον εαυτο της η διεργασία. Εαν ειναι τα acks == 3 τοτε μαρκαρω το μηνυμα σαν έτοιμο. Και κάνω διαγραφη το self.acks[id].

Τέλος κανω ενα προσωρινό αντίγραφο του μηνυματοςπου έλαβα και άν **ΔΕΝ** είναι ack και το ID του αποστολέα **διαφέρει** από το id της διεργασίας μαρκάρει το μήνυμα σαν ack και το κάνει multicast.

def loop(self)
Αυτή η συνάρτηση δημιουργεί ανα 2” για 20 φορές ενα μήνυμα και μέσω της sendUpdate το κάνει multicast. Επίσης μετά απο 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 εάν η **queue** εχει μέσα στοιχεία, κάνει pop το 1ο στοιχείο και **εάν** είναι μαρκαρισμένο σαν **ready** τότε το επιστρέφει σε αυτόν που την κάλεσε, διαφρετικά το επανατοποθετει στην queue.

Τρόπος Εκτέλεσης

Ανοίγουμε σε linux στο ίδιο pc 3 terminal και τρεχουμε με τη σειρά:
1ο: python totormu.py 127.0.0.1 0 2312
2ο: python totormu.py 127.0.0.1 1 2313
3ο: python totormu.py 127.0.0.1 2 2314

Screenshots