Αλγόριθμοι Κατανεμημένων Συστημάτων

Περιεχόμενα

- Εργαλεία Πολυπλοκότητας
- Ο συγχρονισμός στα κατανεμημένα συστήματα
- Το πρόβλημα της εκλογής αρχηγού
- Αμοιβαίος Αποκλεισμός
- Δρομολόγηση
- Έλεγχος τερματισμού
- Ανθεκτικότητα σε λάθη - Σταθεροποίηση

Εργαλεία Πολυπλοκότητας

- $T(n) \in O(f(n))$, αν υπάρχουν σταθερές c και n_0 ώστε $T(n) \leq c*f(n)$, για κάθε $n \geq n_0$
- $T(n) \in \Omega(g(n))$, αν υπάρχουν σταθερές c και n_0 ώστε $T(n) \geq c*g(n)$, για κάθε $n \geq n_0$
- $T(n) \in \Theta(h(n))$, αν $T(n) \in O(h(n))$ και $T(n) \in \Omega(h(n))$

Συγχρονισμός

— Το πρόβλημα του συγχρονισμού στα κατανεμημένα συστήματα γίνεται δυσκολότερο γιατί:

- Οι σχετικές πληροφορίες βρίσκονται συγκεντρωμένες σε πολλές μηχανές
- Οι διεργασίες αποφασίζουν βασιζόμενες σε πληροφορίες που είναι διαθέσιμες τοπικά
- Θα πρέπει να αποφευχθεί η ύπαρξη ενός σημείου ολικής αποτυχίας
- Δεν υπάρχει κοινό ρολόι ή άλλη καθολική πηγή χρόνου

Λογικά Ρολόγια Lamport

- L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", Communications of the ACM, vol. 21, pp. 558-564, 1978.
- Αν δύο διεργασίες δεν αλληλεπιδρούν δεν είναι απαραίτητο να συγχρονιστούν, διότι η έλλειψη συγχρονισμού δεν θα δημιουργήσει πρόβλημα
- Σημασία δεν έχει η συμφωνία για την ώρα, αλλά η χρονική σειρά με την οποία λαμβάνουν χώρα τα γεγονότα.

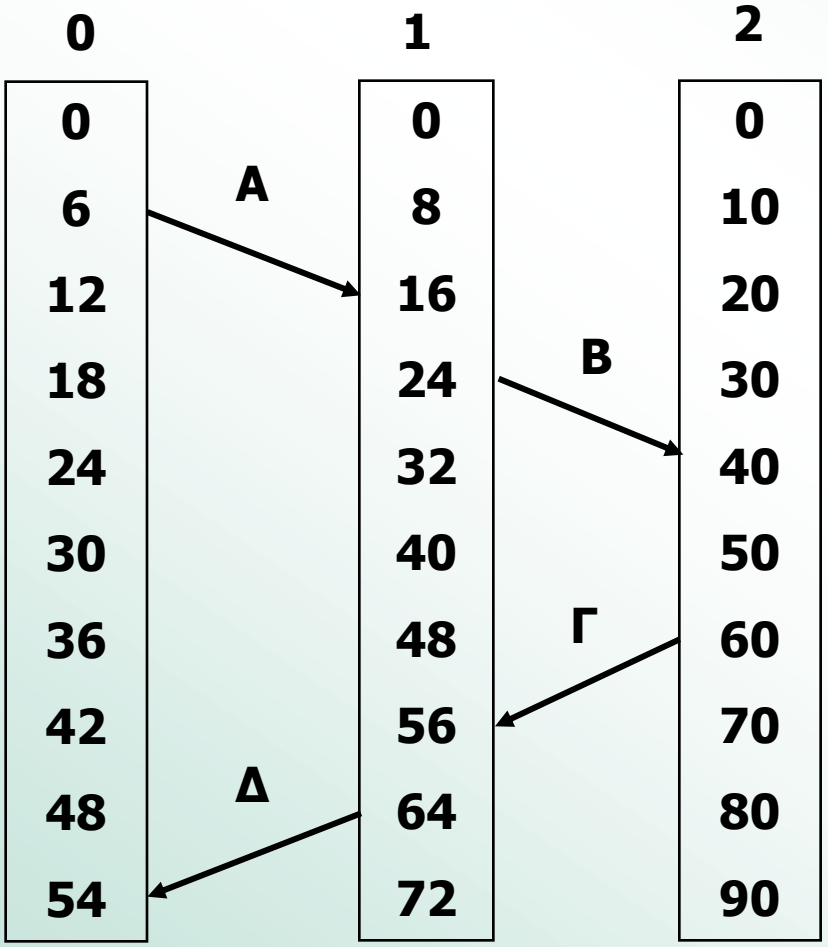
Λογικά Ρολόγια Lamport – Η σχέση «συνέβη πριν»

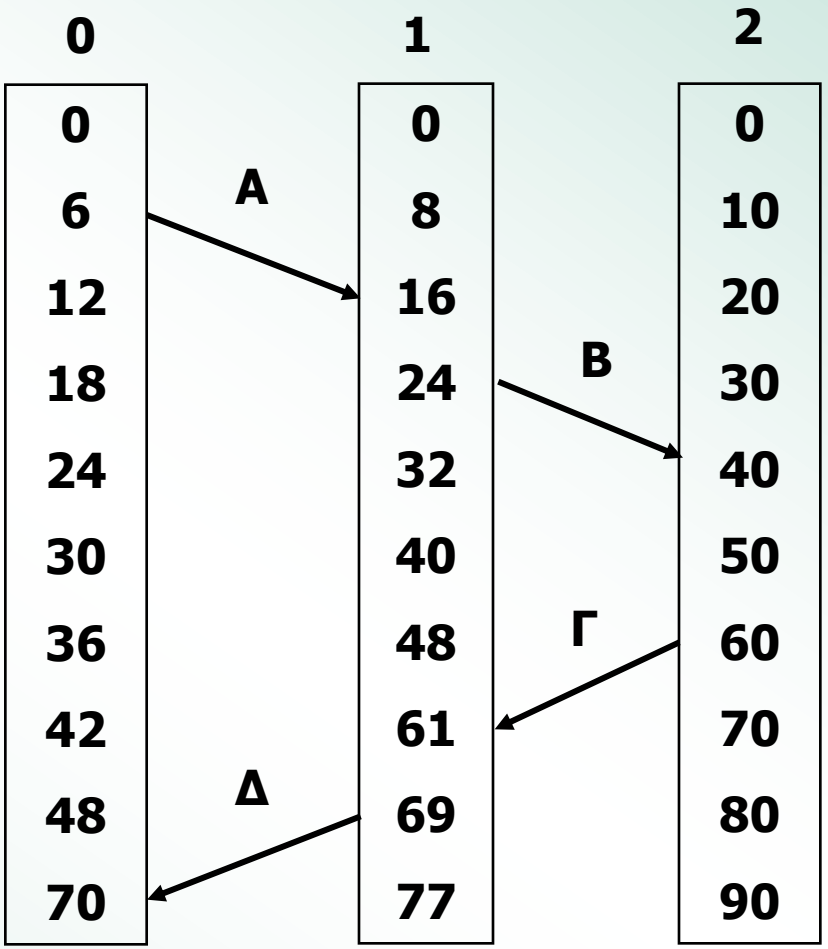
- Η σχέση «συνέβη πριν» ορίζεται ως εξής:
  - Εάν α και β είναι γεγονότα που συμβαίνουν στην ίδια διεργασία και το α συμβαίνει πριν το β, τότε α $\rightarrow$ β
  - Εάν α είναι ένα γεγονός αποστολής μηνύματος από μία διεργασία και β είναι το γεγονός της παραλαβής του αντίστοιχου μηνύματος από άλλη διεργασία, τότε α $\rightarrow$ β
  - Η σχέση «συνέβη πριν» είναι μεταβατική
- Ταυτόχρονα είναι δύο γεγονότα α και β όταν ανήκουν σε διαφορετικές διεργασίες που δεν ανταλλάσσουν μηνύματα (ούτε μέσω τρίτων), και άρα δεν ισχύει α $\rightarrow$ β ούτε β $\rightarrow$ α

Λογικά Ρολόγια Lamport - Επεκτάσεις

- Ανάμεσα σε δύο συμβάντα που συμβαίνουν σε μια διεργασία το ρολόι πρέπει να χτυπήσει τουλάχιστον μια φορά
- Αν δεν θέλουμε να υπάρχουν συμβάντα στο σύστημα που να συμβαίνουν την ίδια χρονική στιγμή, τότε μπορούμε να κατατάξουμε τις διεργασίες με βάση τον αύξοντα αριθμό τους σε υψηλής τάξης και χαμηλής τάξης.

Λογικά Ρολόγια Lamport – Ένα παράδειγμα





Κατανεμημένος Αλγόριθμος (Distributed Algorithm)

- Η συμπεριφορά ενός κατανεμημένου συστήματος προσδιορίζεται από έναν κατανεμημένο αλγόριθμο – έναν ορισμό των βημάτων που εκτελεί κάθε διεργασία που συμμετέχει στο σύστημα καθώς και τα μηνύματα που ανταλλάσσονται μεταξύ των διεργασιών
- Το ρυθμό με τον οποίο εξελίσσεται μια διεργασία και το χρονισμό της μετάδοσης μηνυμάτων μεταξύ τους δεν μπορούμε να τον προβλέψουμε!
- Επίσης δεν είναι εύκολο να περιγράψουμε κάθε φορά την κατάσταση του αλγορίθμου, λόγω αστοχιών διεργασιών και/ή μηνυμάτων

Σύγχρονα και Ασύγχρονα Δίκτυα

- **Σύγχρονα δίκτυα**
  - **Υπάρχει ένα σφαιρικό ρολόι που ακούνε ταυτόχρονα όλοι οι επεξεργαστές**
  - **Αφού δοθεί το σφαιρικό σήμα εκκίνησης η επεξεργασία αρχίζει από όλους τους επεξεργαστές ταυτόχρονα**
  - **Σε κάθε παλμό του ρολογιού κάθε επεξεργαστής εκτελεί ένα υπολογιστικό βήμα και στέλνει μηνύματα**
  - **Υπάρχει εγγύηση ότι όλα τα υπολογιστικά βήματα θα έχουν προλάβει να ολοκληρωθούν και όλα τα μηνύματα θα έχουν φτάσει στον προορισμό τους πριν τον επόμενο παλμό του ρολογιού**

Σύγχρονα και Ασύγχρονα Δίκτυα (συνέχεια)

- **Ασύγχρονα δίκτυα**
  - **Δεν υπάρχει σφαιρικό ρολόι στο σύστημα**
  - **Δεν υπάρχει σφαιρικό σήμα εκκίνησης**
  - **Οι καθυστερήσεις διάδοσης των μηνυμάτων μεταξύ των επεξεργαστών είναι απρόβλεπτες**
  - **Κάθε μήνυμα φτάνει μετά από πεπερασμένο αλλά απροσδιόριστο χρόνο στον προορισμό του**

Συγχρονιστές

- Πρωτόκολλα που προσομοιώνουν ένα ιδεατό σύγχρονο περιβάλλον πάνω από ένα ασύγχρονο δίκτυο
- Πρόκειται για ένα ασύγχρονο κατανεμημένο πρωτόκολλο που τρέχει στο ασύγχρονο σύστημα παράλληλα με ένα σύγχρονο αλγόριθμο του οποίου η εκτέλεση είναι ο τελικός στόχος
- Κάθε κόμβος θα πρέπει στο πέρας του βήματος i του σύγχρονου αλγόριθμου να μην προχωρά στον επόμενο χτύπο ρολογιού αν δεν έχει λάβει όλα τα αποτελέσματα του βήματος i των γειτόνων του

Ο συγχρονιστής α του Awerbuch

- Ένας κόμβος είναι ασφαλής ως προς ένα συγκεκριμένο παλμό αν κάθε μήνυμα που έχει στείλει στον παλμό αυτό έχει φτάσει στον προορισμό του. Η αναγνώριση γίνεται με αποστολή μηνύματος ack από τους γείτονες
- Κάθε κόμβος όταν μαθαίνει ότι είναι ασφαλής το αναφέρει σε όλους τους γείτονές του με μηνύματα
- Όταν ένας κόμβος μάθει πως όλοι οι γείτονές του είναι ασφαλείς προχωρά στο επόμενο βήμα
- Είναι αποδοτικός ως προς το χρόνο, αλλά όχι ως προς τα μηνύματα

Ο συγχρονιστής α του Awerbuch: ανάλυση πολυπλοκότητας

- Έστω ότι V είναι ο πληθάριθμος του συνόλου των κορυφών (κόμβων) στο γράφο, και E ο πληθάριθμος του αντίστοιχου συνόλου των ακμών
- Από κάθε ακμή περνάνε ακριβώς δύο μηνύματα (ένα προς κάθε κατεύθυνση) ανά παλμό. Άρα σε κάθε παλμό έχουμε 2E μηνύματα, δηλαδή O(E) μηνύματα ανά παλμό ρολογιού
- Κάθε γράφος V κόμβων έχει το πολύ V(V-1)/2 ακμές (πλήρης γράφος), άρα έχουμε $O(V^2)$ μηνύματα
- Οι καθυστερήσεις των μηνυμάτων είναι 1 χρονική μονάδα. Αφού οι επικοινωνίες σε κάθε παλμό γίνονται μεταξύ γειτονικών κόμβων, κάθε κόμβος σε σταθερό χρόνο καταλαβαίνει ότι είναι ασφαλής. Άρα για κάθε παλμό έχουμε χρόνο O(1)

Ο συγχρονιστής β του Awerbuch

- Απαιτείται μια αρχική φάση κατά την οποία εκλέγεται ένας αρχηγός στο δίκτυο και κατασκευάζεται ένα γεννητικό δέντρο με ρίζα τον κόμβο αρχηγό
- Ο αρχηγός δίνει το σήμα ενεργοποίησης του επόμενου βήματος μέσω του δέντρου
- Μόλις ένας κόμβος γνωρίζει ότι ο ίδιος είναι ασφαλής και το ίδιο συμβαίνει για όλα τα παιδιά του στο δέντρο, το αναφέρει στον πατέρα του
- Είναι αποδοτικός σε μηνύματα, αλλά όχι σε χρόνο

Ο συγχρονιστής β του Awerbuch: ανάλυση πολυπλοκότητας

- Το γεννητικό δένδρο που κατασκευάζεται έχει V-1 ακμές. Σε κάθε παλμό έχουμε 2 μηνύματα ανά ακμή
- Άρα κινούνται O(V) μηνύματα ανά παλμό στη χειρότερη περίπτωση
- Ο χρόνος που απαιτείται εξαρτάται από το ύψος του δένδρου
- Το ύψος του δένδρου είναι το πολύ V-1, άρα στη χειρότερη περίπτωση έχουμε O(V) χρονικές μονάδες

Ο συγχρονιστής γ του Awerbuch

- Είναι συνδυασμός των α και β συγχρονιστών
- Το σύνολο των κόμβων χωρίζεται σε ξένα μεταξύ τους υποσύνολα (clusters). Σε κάθε cluster εκλέγεται ένας αρχηγός και δημιουργείται ένα γεννητικό δέντρο με ρίζα τον αρχηγό
- Για κάθε δύο γειτονικά clusters επιλέγεται ένα link διασύνδεσης (preferred link)
- Στην πρώτη φάση του γ εκτελείται σε κάθε cluster ο αλγόριθμος β
- Στη δεύτερη φάση του αλγορίθμου οι κόμβοι ενός cluster περιμένουν μέχρι να καταστούν ασφαλείς όλοι οι γειτονικοί clusters (αλγόριθμος α)
- Όταν όλοι οι γειτονικοί ενός cluster καταστούν ασφαλείς τότε ενεργοποιείται ο επόμενος παλμός

Ο συγχρονιστής γ του Awerbuch

- **Τύποι μηνυμάτων που ανταλλάσσονται:**
  - **PULSE: ενεργοποιεί ένα νέο παλμό στους κόμβους ενός cluster**
  - **SAFE: ένας κόμβος δηλώνει ότι είναι ασφαλής**
  - **CLUSTER_SAFE: δηλώνει ότι όλος ο cluster είναι ασφαλής (διαδίδεται σε όλο το cluster και στα γειτονικά clusters μέσω των preferred links)**
  - **READY: δηλώνει ότι ένας κόμβος είναι έτοιμος για τον επόμενο παλμό (διαδίδεται από τα φύλλα προς τον αρχηγό σε ένα cluster όταν ο κόμβος έχει λάβει όλα τα απαραίτητα μηνύματα READY από τα παιδιά του και όλα τα απαραίτητα CLUSTER_SAFE από τα preferred links που καταλήγουν σε αυτόν.**

Ο συγχρονιστής γ του Awerbuch: ανάλυση πολυπλοκότητας

- Έστω V το σύνολο των κόμβων του δικτύου, E το σύνολο των links που είτε ανήκουν σε κάποιο δένδρο είτε είναι preferred links, και H το μέγιστο ύψος από όλα τα δένδρα
- Από κάθε link δένδρου περνάνε σε κάθε παλμό οπωσδήποτε τα μηνύματα PULSE, SAFE, CLUSTER_SAFE και READY μία φορά το καθένα
- Από κάθε preferred link περνούν τα μηνύματα CLUSTER_SAFE και READY από μία φορά το καθένα
- Άρα έχουμε O(E) μηνύματα ανά παλμό ρολογιού.
- Σε κάθε cluster ο χρόνος ενός παλμού είναι O(H)
- Ο χρόνος διάδοσης των CLUSTER_SAFE μηνυμάτων είναι O(H). Άρα ο συνολικός χρόνος είναι O(H).

Αλγόριθμοι εκλογής αρχηγού

- Πολλοί κατανεμημένοι αλγόριθμοι απαιτούν την ύπαρξη ενός αρχηγού ή συντονιστή στο σύστημα
- Θεωρούμε ότι κάθε διεργασία διαθέτει ένα αριθμό που την διακρίνει μοναδικά
- Ο αλγόριθμος προσπαθεί να εξασφαλίσει ότι όλες οι διεργασίες θα συμφωνήσουν σε ένα αρχηγό

Ο αλγόριθμος των Garcia-Molina (αλγόριθμος του ισχυρότερου)

- Υποθέτουμε ότι κάθε διεργασία γνωρίζει τους αριθμούς όλων των άλλων διεργασιών, αλλά δεν γνωρίζει ποιες από αυτές λειτουργούν κανονικά
- Όταν μια διεργασία P αντιληφθεί ότι υπάρχει ανάγκη για ένα συντονιστή στέλνει ένα μήνυμα σε όλες τις διεργασίες με μεγαλύτερο αριθμό
- Αν δεν πάρει απάντηση η P ανακηρύσσεται αρχηγός
- Αν κάποια απαντήσει τότε αναλαμβάνει να συνεχίσει τη διαδικασία εκλογής
- Η διεργασία που θα εκλεγεί ανακοινώνει την εκλογή

Αλγόριθμος Δένδρου

- Είναι αλγόριθμος κύματος.
- Θεωρούμε ότι η τοπολογία είναι δένδρο ή οποιαδήποτε αυθαίρετη τοπολογία για την οποία όμως ένα γεννητικό δένδρο είναι διαθέσιμο.
- Υποθέτουμε ότι όλα τα φύλλα του δένδρου εκκινούν τον αλγόριθμο.
- Κάθε διεργασία στέλνει ακριβώς ένα μήνυμα στον αλγόριθμο.
- Εάν μία διεργασία έχει λάβει ένα μήνυμα από κάθε γείτονά της εκτός ενός, η διεργασία στέλνει ένα μήνυμα προς τον υπολειπόμενο γείτονα.
- Εάν μία διεργασία έχει λάβει μήνυμα από όλους τους γείτονές της, αποφασίζει.

Αλγόριθμος Δένδρου

- Είναι απαραίτητο τουλάχιστον όλα τα φύλλα να είναι αρχικοποιητές του αλγορίθμου.
- Προστίθεται μία φάση αφύπνισης (wakeup phase).
- Η διεργασία που θέλει να ξεκινήσει την εκλογή διοχετεύει στο δένδρο ένα μήνυμα wakeup με σκοπό αυτό να φτάσει σε όλες τις διεργασίες.
- Όταν μία διεργασία έχει λάβει ένα μήνυμα wakeup από κάθε κανάλι επικοινωνίας (άρα από όλους τους γείτονές της), ξεκινά τον αλγόριθμο του διάδοσης του δένδρου για τον υπολογισμό της μικρότερης ταυτότητας.
- Όταν μία διεργασία αποφασίσει γνωρίζει την ταυτότητα του αρχηγού.
- Εάν αυτή η ταυτότητα ισούται με την ταυτότητά της, γίνεται αρχηγός, αλλιώς χάνει την εκλογή.

Αλγόριθμος Δένδρου

```
var     ws: boolean init false; wr: integer  init 0;  rec[q]: Boolean for each q ∈ Neigh    init false;
        v: P init p;   state: (sleep, leader, lost) init sleep;
begin if p is initiator then
        begin ws:=true;
          forall q ∈ Neigh do send <wakeup> to q
        end;
        while wr < #Neigh do
          begin receive <wakeup>;
                    wr:=wr+1;
                    if not ws then
                              begin ws:=true;
                                        forall q ∈ Neigh do send <wakeup> to q
                              end
          end;
```

Αλγόριθμος Δένδρου

```
/* Now start the tree algorithm */
        while #{q: ¬rec[q]} >1 do
          begin receive <tok,r> from q;
                    rec[q]:=true;
                    v:=min(v,r)
          end;
        send <tok, v> to q0 with ¬rec[q0];
        receive <tok,r> from q0;
        v:=min(v,r);
        if v=p then state:=leader else state:=lost;
        forall q ∈ Neigh, q≠q0 do send <tok,v> to q
end
```

Αλγόριθμος Δένδρου - Πολυπλοκότητα

- Από κάθε ακμή του δένδρου στέλνονται δύο <wakeup> και δύο <tok,r> μηνύματα. Αφού ο αριθμός των ακμών στο δένδρο είναι N-1, N ο αριθμός των κόμβων, τότε η πολυπλοκότητα των μηνυμάτων είναι 4N-4, δηλαδή O(N).
- Μέσα σε D χρονικές μονάδες από τη στιγμή που ξεκινά τον αλγόριθμο η πρώτη διεργασία, D το ύψος του δένδρου, κάθε διεργασία έχει στείλει <wakeup> μηνύματα, έτσι μέσα σε D+1 χρονικές μονάδες κάθε διεργασία έχει ξεκινήσει τον αλγόριθμο κύματος.
- Είναι εύκολο να διαπιστώσουμε ότι η πρώτη απόφαση λαμβάνει χώρα το πολύ D χρονικές μονάδες μετά την εκκίνηση του κύματος και η τελευταία το πολύ D χρονικές μονάδες μετά την πρώτη απόφαση. Άρα συνολικά έχουμε 3D+1 χρονικές μονάδες, δηλαδή O(D).

Ο αλγόριθμος δακτυλίου του LeLann

- Υποθέτουμε ότι οι κόμβοι του συστήματος είναι οργανωμένοι σε μορφή δακτυλίου
- Υποθέτουμε ότι κάθε διεργασία γνωρίζει τον αριθμό της, αλλά όχι των άλλων. Γνωρίζει όμως ποια είναι η επόμενή της στο δακτύλιο
- Οι διεργασίες που ξεκινούν τον αλγόριθμο (αρχικοποιητές) στέλνουν στην επόμενή τους ένα μήνυμα με τον αριθμό τους (token)
- Όταν μια διεργασία λάβει πίσω το μήνυμά της σημαίνει ότι έχει λάβει και όλα τα μηνύματα των άλλων αρχικοποιητών
- Εκλέγεται από τους αρχικοποιητές η διεργασία με τον μικρότερο αριθμό

Αλγόριθμος LeLann: Ανάλυση Πολυπλοκότητας

- Ο αλγόριθμος του LeLann λύνει το πρόβλημα εκλογής αρχηγού σε δακτύλιο χρησιμοποιώντας $O(N^2)$ μηνύματα και σε $O(N)$ χρόνο στη χειρότερη περίπτωση. **Απόδειξη**:
- Υπάρχουν το πολύ N διαφορετικά tokens που κυκλοφορούν στο δακτύλιο, όπου το καθένα κάνει N βήματα. Άρα η πολυπλοκότητα μηνυμάτων είναι $O(N^2)$
- Σε N-1 χρονικές μονάδες το πολύ από τη στιγμή που ο πρώτος αρχικοποιητής έχει στείλει το token του, κάθε άλλος αρχικοποιητής έχει στείλει το δικό του, και κάθε αρχικοποιητής λαμβάνει το δικό του token N χρονικές μονάδες μετά. Άρα ο αλγόριθμος τερματίζει μετά από το πολύ 2N-1 χρονικές μονάδες.

Ο αλγόριθμος δακτυλίου των Chang & Roberts

- Βελτιώνει τον αλγόριθμο του LeLann σε αριθμό μηνυμάτων στην μέση περίπτωση και σε χρόνο στην χειρότερη περίπτωση
- Ένας αρχικοποιητής δεν προωθεί το μήνυμα μιας διεργασίας στο δακτύλιο αν καταλάβει ότι αυτή χάνει την εκλογή
- Εκλέγεται ο αρχικοποιητής που θα λάβει το δικό του μήνυμα

Αλγόριθμος Chang & Roberts: Ανάλυση Πολυπλοκότητας

- Ο αλγόριθμος των Chang & Roberts λύνει το πρόβλημα εκλογής αρχηγού σε δακτύλιο χρησιμοποιώντας $\Theta(N^2)$ μηνύματα στη χειρότερη περίπτωση. **Απόδειξη**:
- Χρησιμοποιούνται το πολύ N tokens, όπου κάθε token κάνει το πολύ N βήματα. Άρα έχουμε $O(N^2)$ μηνύματα
- Υποθέτουμε ακόμη ότι οι κόμβοι τοποθετούνται με αύξουσα σειρά πάνω στο δακτύλιο, οπότε όλα τα tokens «κόβονται» από τον κόμβο 0, πλην του δικού του.
- Άρα κάθε token i προωθείται κατά N-i βήματα. Άρα το σύνολο των μηνυμάτων είναι (1/2)N(N+1). Άρα έχουμε $\Omega(N^2)$ μηνύματα

Αλγόριθμος Chang & Roberts: Ανάλυση Πολυπλοκότητας

- Ο αλγόριθμος των Chang & Roberts απαιτεί O(N logN) μηνύματα στη μέση περίπτωση, θεωρώντας ότι όλοι οι κόμβοι είναι αρχικοποιητές.
- Απόδειξη:
  - Θα υπολογίσουμε το μέσο αριθμό μηνυμάτων από όλες τις διαφορετικές κυκλικές διατάξεις των N κόμβων
  - Έστω ότι είναι s ο κόμβος με τη μικρότερη ταυτότητα και p_i ο κόμβος που βρίσκεται i βήματα πριν τον s στο δακτύλιο
  - Υπάρχουν (N-1)! διαφορετικές διατάξεις στο δακτύλιο (εξαιρείται ο s)
  - Θα υπολογίσουμε τον αριθμό των μηνυμάτων του p_i για όλες τις διατάξεις και θα αθροίσουμε για όλα τα i

Αλγόριθμος Chang & Roberts: Ανάλυση Πολυπλοκότητας

- Το μήνυμα του s θα περάσει N φορές σε κάθε διάταξη και άρα N(N-1)! φορές συνολικά
- Το μήνυμα του i θα περάσει από i κόμβους το πολύ πριν το «κόψει» ο s
- Έστω $A_{i,k}$ ο αριθμός των κυκλικών διατάξεων όπου το μήνυμα του p_i περνάει ακριβώς k φορές. Ο συνολικός αριθμός των μηνυμάτων του p_i είναι

$$\sum_{k=1}^{i} (k \times A_{i,k})$$

- Το μήνυμα του p_i περνάει ακριβώς i φορές στο (1/i)(N-1)! των διατάξεων
- Παρόμοια, το ίδιο μήνυμα περνάει ακριβώς k φορές (k≤i) στο (1/k)(N-1)! των διατάξεων
- Αυτό σημαίνει ότι το μήνυμα θα περάσει τουλάχιστον k φορές, αλλά όχι τουλάχιστον k+1 φορές. Οπότε ο αριθμός των διατάξεων που αυτό συμβαίνει είναι (για k<i):

Αλγόριθμος Chang & Roberts: Ανάλυση Πολυπλοκότητας

$$A_{i,k} = \frac{1}{k}(N-1)! - \frac{1}{k+1}(N-1)! = \frac{1}{k(k+1)}(N-1)!$$

- Ο συνολικός αριθμός λοιπόν είναι:

$$\sum_{k=1}^{i-1} k\left(\frac{1}{k(k+1)}(N-1)!\right) + i\frac{1}{i}(N-1)! = \left(\sum_{k=1}^{i}\frac{1}{k}\right)(N-1)!$$

- Ο πρώτος όρος στο παραπάνω γινόμενο λέγεται i-οστός αρμονικός αριθμός και συμβολίζεται H_i. Ισχύει η ταυτότητα:

$$\sum_{i=1}^{m} H_i = (m+1)H_m - m$$

Αλγόριθμος Chang & Roberts: Ανάλυση Πολυπλοκότητας

- Αθροίζοντας λοιπόν όλα τα περάσματα των μηνυμάτων για να πάρουμε το συνολικό αριθμό και εφαρμόζοντας την παραπάνω ταυτότητα έχουμε ότι:

$$\sum_{i=1}^{N-1}[H_i(N-1)!] = (N \times H_{N-1} - (N-1)) \times (N-1)!$$

- Αθροίζοντας τα N*(N-1)! περάσματα φτάνουμε σε ένα τελικό αριθμό περασμάτων $(N*H_{N-1}+1)(N-1)!=(N*H_N)(N-1)!$
- Ισχύει ότι $(N*H_N) \cong 0.69NlogN$

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh

- Ο αλγόριθμος αυτός επιτυγχάνει πολυπλοκότητα μηνυμάτων O(NlogN) στη χειρότερη περίπτωση βελτιώνοντας τον αλγόριθμο των Chang & Roberts.
- Αρχικά κάθε ταυτότητα είναι ενεργή, αλλά σε κάθε γύρο κάποιες ταυτότητες γίνονται παθητικές. Σε ένα γύρο μία ενεργή ταυτότητα συγκρίνει τον εαυτό της με τις ενεργές ταυτότητες των δύο γειτονικών της κόμβων.
- Έστω ότι εκλέγεται η μικρότερη ταυτότητα, αν μία ενεργή ταυτότητα αποτελεί τοπικό ελάχιστο, τότε προχωρά στον επόμενο γύρο, διαφορετικά γίνεται παθητική. Έτσι μετά από logN γύρους θα έχει απομείνει στο δακτύλιο μόνο μία ενεργή ταυτότητα και αυτή κερδίζει την εκλογή.

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh

- Η ιδέα αυτή δεν μπορεί να εφαρμοστεί απ' ευθείας σε δακτυλίους που είναι μονής κατεύθυνσης.
- Ας υποθέσουμε ότι έχουμε ένα δακτύλιο όπου βρίσκονται μεταξύ άλλων τρεις ενεργές σε σειρά ταυτότητες r,q,p. Τότε η q μπορεί να λάβει το μήνυμα της r, αλλά δεν μπορεί να λάβει το μήνυμα της p γιατί ο δακτύλιος είναι μονής κατεύθυνσης.
- Για να γίνει όμως η σύγκριση η q στέλνει την ταυτότητά της στην p και η r δεν στέλνει απλώς την ταυτότητά της στην q, αλλά η q την προωθεί στην p, ώστε η p να κάνει τη σύγκριση.

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh

- Οι διεργασίες που χάνουν την εκλογή και γίνονται παθητικές, απλώς προωθούν τα μηνύματα που λαμβάνουν.
- Εάν μία διεργασία λάβει μία ταυτότητα που είναι ίση με τη δική της τότε ανακηρύσσεται αρχηγός
- Αν λάβει μία ταυτότητα διαφορετική από τη δική της συγκρίνει την ταυτότητα αυτή με τη δική της και με το τρέχον ελάχιστο που έχει κρατήσει από τις προηγούμενες συγκρίσεις.
- Αν η παραληφθείσα ταυτότητα είναι η μικρότερη, τότε η διεργασία γίνεται παθητική και κρατά το νέο ελάχιστο.

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh

```
var     ci_p : P init p; /* Current identity of p */  acn_p : P init undefined; /* id of anticlockwise active neighbor */
        win_p : P init undefined; /* id of winner */ state_p : (active, passive, leader, lost) init active;

begin
        if p is initiator then state_p:= active else state_p:=passive;
        while win_p=undefined do {
            if state_p=active then {
                        send <one,ci_p>; receive <one,q>; acn_p:=q;
                        if acn_p=ci_p then  {/* acn_p is the minimum */
                                    send <smal,acn_p>; win_p:=acn_p; receive <smal,q>;
                        }
                        else { /* acn_p is current id of neighbor */
                                    send <two,acn_p>; receive <two,q>;
                                    if acn_p < ci_p and acn_p < q then ci_p:=acn_p;
                                    else state_p:=passive;
                        }
            }
            else { /* state_p=passive */
                        receive <one,q>; send <one,q>;  receive m;
                        send m; /* m is either <two,q> or <smal, q> */
                        if m is a <smal,q> message then win_p:=q
            }
        }
        if p=win_p then state_p:=leader else state_p:=lost
}
```

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh - Πολυπλοκότητα

- Λέμε ότι μία διεργασία βρίσκεται στον γύρο i όταν εκτελεί τη βασική επανάληψη του αλγορίθμου για i-οστή φορά.
- Οι γύροι δεν είναι σφαιρικά συγχρονισμένοι. Είναι πιθανό μία διεργασία να βρίσκεται αρκετούς γύρους μπροστά από μία άλλη διεργασία που βρίσκεται σε άλλο τμήμα του δακτυλίου.
- Αλλά, αφού κάθε διεργασία στέλνει και λαμβάνει ακριβώς δύο μηνύματα σε κάθε γύρο και τα κανάλια είναι FIFO, ένα μήνυμα λαμβάνεται συνεχώς στον ίδιο γύρο που στέλνεται.
- Στον πρώτο γύρο όλοι οι αρχικοποιητές είναι ενεργοί και κάθε ενεργή διεργασία διατηρεί μια διαφορετική «τρέχουσα ταυτότητα».

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh - Πολυπλοκότητα

- Πρέπει να δείξουμε ότι εάν ο γύρος i ξεκινά με k>1 ενεργές διεργασίες και κάθε διεργασία διατηρεί ένα διαφορετικό ci, τότε τουλάχιστον μία και το πολύ k/2 διεργασίες επιβιώνουν στο γύρο.
- Με την ανταλλαγή των <one, q> μηνυμάτων, τα οποία διαδίδονται επίσης από τις παθητικές διεργασίες, κάθε ενεργή διεργασία αποκτά την τρέχουσα ταυτότητα του πρώτου ενεργού γείτονα προς την αντίθετη φορά, η οποία σε όλες τις περιπτώσεις είναι διαφορετική από τη δική της.
- Ακολούθως, κάθε ενεργή διεργασία συνεχίζει το γύρο με την ανταλλαγή των <two,q> μηνυμάτων, με τα οποία κάθε ενεργή διεργασία αποκτά την τρέχουσα ταυτότητα του δεύτερου ενεργού γείτονα προς την αντίθετη φορά.
- Κάθε ενεργή διεργασία πλέον κατέχει μια διαφορετική τιμή για τη μεταβλητή acn, η οποία υπονοεί ότι οι επιζώντες του γύρου έχουν όλοι διαφορετική ταυτότητα στο τέλος του γύρου.

$\lfloor \log N \rfloor + 1$

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh - Πολυπλοκότητα

- Τουλάχιστον η ταυτότητα που ήταν η μικρότερη στην αρχή του γύρου επιβιώνει, έτσι υπάρχει τουλάχιστον μία διεργασία που επέζησε. Μία ταυτότητα επόμενη σε ένα τοπικό ελάχιστο δεν είναι τοπικό ελάχιστο, που σημαίνει ότι ο αριθμός των επιζώντων είναι το πολύ k/2.
- Επίσης, αυτό σημαίνει ότι θα υπάρχει ένας γύρος με αριθμό ≤ $\lfloor \log N \rfloor + 1$

  που ξεκινά με ακριβώς μία ενεργή ταυτότητα, που είναι η μικρότερη ταυτότητα κάποιου αρχικοποιητή.
- Εάν ένας γύρος ξεκινήσει με ακριβώς μία ενεργή διεργασία p, με τρέχουσα ταυτότητα ci_p, ο αλγόριθμος ολοκληρώνεται μετά από αυτό το γύρο με $win_p=ci_p$ για κάθε q. Κι αυτό συμβαίνει γιατί το μήνυμα <one,ci_p> της p διεργασίας διαδίδεται από όλες τις διεργασίες και τελικά λαμβάνεται από την ίδια την p. Η διεργασία p αποκτά $acn_p=ci_p$ και στέλνει ένα μήνυμα <smal,acn_p> στο δακτύλιο, το οποίο προκαλεί κάθε διεργασία q να βγει από τη βασική επανάληψη με $win_p=acn_p$

Ο αλγόριθμος των Peterson/Dolev-Klawe-Rodeh - Πολυπλοκότητα

- Υπάρχουν λοιπόν το πολύ $\lfloor \log N \rfloor + 1$ γύροι, σε κάθε ένα από τους οποίους ανταλλάσσονται ακριβώς 2N μηνύματα, το οποίο αποδεικνύει ότι η πολυπλοκότητα μηνυμάτων φράσσεται από το 2NlogN+O(N).

Ο αλγόριθμος των Itai & Rodeh

- Θεωρούμε ένα δακτύλιο ανωνύμων κόμβων
- Δηλαδή είναι πολύ πιθανό να υπάρχουν επεξεργαστές με την ίδια μεγαλύτερη (ή μικρότερη) ταυτότητα στο δίκτυο.
- Ο αλγόριθμος κινείται στα πλαίσια εκείνου των Chang & Roberts με την κύρια διαφορά ότι οι επεξεργαστές δεν είναι αριθμημένοι με μοναδικό τρόπο.
- Αρχικά οι επεξεργαστές διαλέγουν για ταυτότητές τους κάποιους τυχαίους αριθμούς από το σύνολο {1,...,n} και στη συνέχεια γίνεται η εκλογή του μεγαλύτερου με ταυτόχρονο έλεγχο της μοναδικότητάς του.

Ο αλγόριθμος των Itai & Rodeh

- Αν κάτι τέτοιο δεν είναι αλήθεια ο αλγόριθμος επαναλαμβάνεται.
- Για τον έλεγχο της μοναδικότητας είναι απαραίτητη η γνώση από όλους τους επεξεργαστές του μεγέθους του δακτυλίου
- Οι επαναλήψεις του αλγορίθμου σημειώνονται από τον αριθμό της φάσης που όπως θα δούμε μεταφέρεται και από τα μηνύματα του αλγορίθμου.
- Σε κάθε φάση του αλγορίθμου βγαίνουν από το παιχνίδι της εκλογής οι επεξεργαστές που καταλαβαίνουν πως είναι αδύνατον να εκλεγούν, επειδή ακριβώς μαθαίνουν πως υπάρχει ισχυρότερος υποψήφιος.
- Αν στη συγκεκριμένη φάση δεν υπάρξει μοναδικός νικητής συνεχίζουν μόνο οι επεξεργαστές που παρέμειναν ενεργοί στο τέλος της προηγούμενης φάσης.

Ο αλγόριθμος των Itai & Rodeh

- Ο επεξεργαστής δεν μπορεί βασισμένος μόνο στην ταυτότητα που κουβαλάει μαζί του ένα μήνυμα να διακρίνει με σιγουριά τα δικά του μηνύματα, αφού η αρίθμηση των επεξεργαστών δεν είναι εγγυημένα μοναδική.
- Κάθε μήνυμα συνοδεύεται από ένα μετρητή που έχει την τιμή 1 στο ξεκίνημα του μηνύματος και αυξάνεται κάθε φορά που το μήνυμα προωθείται από κάποιον επεξεργαστή.
- Ένας επεξεργαστής που θα δεχτεί το μήνυμα με το μετρητή του ίσο με N γνωρίζει με βεβαιότητα πως είναι δικό του μήνυμα που συμπλήρωσε κύκλο και επέστρεψε σε αυτόν.

Ο αλγόριθμος των Itai & Rodeh

var $state_p$: (sleep, cand, leader, lost) init sleep;

$level_p$:integer init 0;

id_p: integer;

$stop_p$:boolean init false;

if p is initiator then {

$level_p$:=1; $state_p$:=cand; id_p:=rand({1,…,N});

send <tok, $level_p$, id_p, 1, true> to $Next_p$

}

Ο αλγόριθμος των Itai & Rodeh

```
while not stop_p do {
        receive a message;
        if it is a token <tok, level, id, hops, un> then
            if hops=N and state_p=cand then
                        if un then {
                                    state_p:=leader; send <ready> to Next_p; receive <ready>; stop_p:=true;
                        }
                        else {
                                    level_p:=level_p+1; id_p:=rand({1,…,N}); send <tok,level_p,id_p,1,true> to Next_p
                        }
            else if level>level_p or (level=level_p and id < id_p) then {
                        level_p:=level; state_p:=lost; send <tok, level, id, hops+1, false> to Next_p
            }
            else if level=level_p and id=id_p then
                        send <tok, level, id, hops+1, false> to Next_p
            else skip /* purge the token */
        else {
            send <ready> to Next_p; stop_p:=true;
        }
}
```

Αμοιβαίος Αποκλεισμός: Συγκεντρωτικός Αλγόριθμος

- Μία διεργασία επιλέγεται ως συντονιστής
- Όταν μία διεργασία θέλει να εισέλθει στην κρίσιμη περιοχή ρωτά τον συντονιστή.
- Αν καμία άλλη διεργασία δεν βρίσκεται στην κρίσιμη περιοχή, τότε ο συντονιστής δίνει άδεια εισόδου.
- Διαφορετικά:
  - Αφήνει τη διεργασία να περιμένει χωρίς απάντηση ή
  - Στέλνει αρνητική απάντηση
- Όταν μια κρίσιμη περιοχή ελευθερώνεται τότε ο συντονιστής επιλέγει την πρώτη διεργασία στην ουρά.

Αμοιβαίος Αποκλεισμός: Κατανεμημένος Αλγόριθμος των Ricart & Agrawala

- Ο αλγόριθμος απαιτεί την ολική διάταξη των γεγονότων στο σύστημα (π.χ. λογικά ρολόγια Lamport)
- Όταν μία διεργασία επιθυμεί την είσοδό της σε μία κρίσιμη περιοχή στέλνει ένα μήνυμα σε όλες τις άλλες
- Όταν μία διεργασία παραλήπτης λαμβάνει ένα τέτοιο μήνυμα:
  - Αν δεν βρίσκεται στην κρίσιμη περιοχή και δεν επιθυμεί να εισέλθει απαντά OK
  - Αν βρίσκεται στην κρίσιμη περιοχή
    - Δεν απαντά (με όριο χρόνου αναμονής στη μεριά του αποστολέα)
    - Στέλνει αρνητική απάντηση
  - Αν θέλει να εισέλθει στην κρίσιμη περιοχή συγκρίνει τις χρονοσφραγίδες και απαντά αναλόγως
- Όταν μία διεργασία αποστολέας μαζέψει όλες τις θετικές απαντήσεις εισέρχεται στην κρίσιμη περιοχή.

Αμοιβαίος Αποκλεισμός: Κατανεμημένος Αλγόριθμος token ring

- Οι διεργασίες τοποθετούνται σε λογική σειρά μορφής δακτυλίου
- Ένα token γυρίζει στο δακτύλιο
- Η διεργασία που κατέχει το token μπορεί να εισέλθει στην κρίσιμη περιοχή αν το επιθυμεί
- Αν δεν το επιθυμεί απλώς προωθεί το token στην επόμενή της στο δακτύλιο διεργασία.

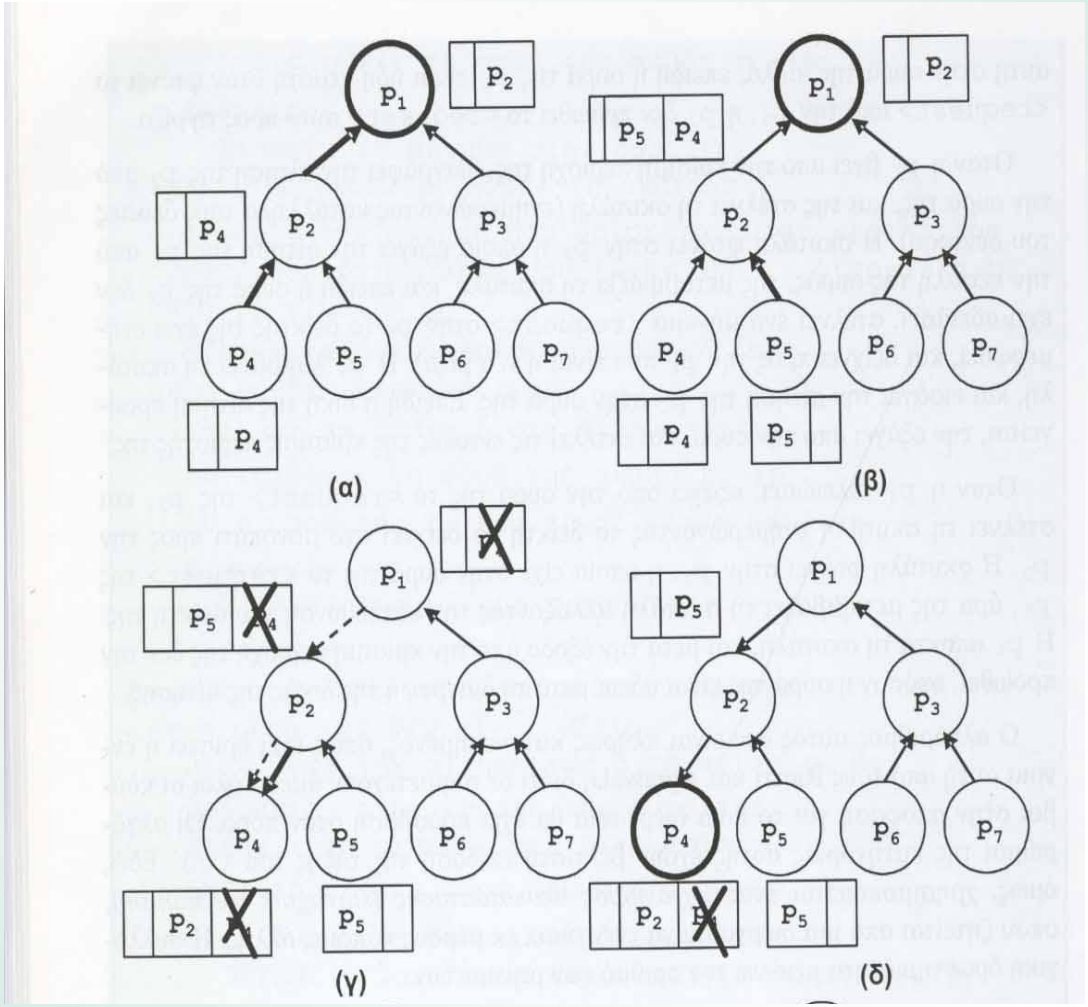
Αμοιβαίος Αποκλεισμός: Αλγόριθμος Raymond

- Οι διεργασίες οργανώνονται σε ένα logical spanning tree
- Το πλήθος των μηνυμάτων που ανταλλάσσονται είναι O(logN).
- Οι ακμές του δένδρου αυτού είναι κατευθυνόμενες και δείχνουν πάντα στον πατέρα του κόμβου, από τον οποίο ξεκινάνε.
- Το δικαίωμα εισόδου στην κρίσιμη περιοχή το έχει μία μόνο διεργασία, ο κάτοχος (holder) της σκυτάλης.
- Η διεργασία αυτή βρίσκεται πάντα στη ρίζα του δένδρου.
- Η σκυτάλη διασχίζει τις ακμές του δένδρου στο μονοπάτι από τη ρίζα μέχρι τη διεργασία που το ζήτησε.

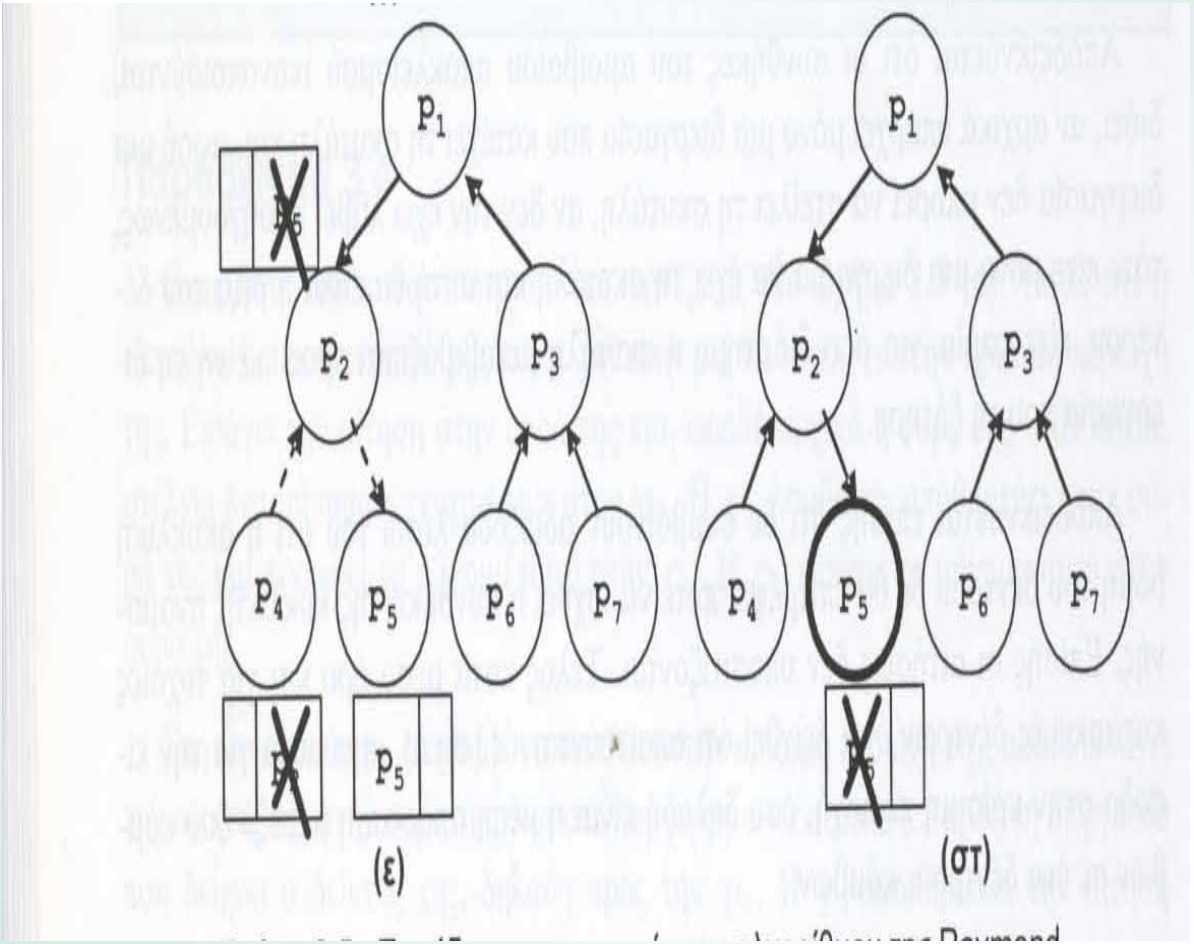
Αμοιβαίος Αποκλεισμός: Αλγόριθμος Raymond

- Καθώς μεταβιβάζεται η σκυτάλη, η κατεύθυνση των ακμών αλλάζει κατά μήκος του μονοπατιού, προκειμένου στη νέα ρίζα να βρίσκεται η διεργασία που ζήτησε τη σκυτάλη.
- Κάθε διεργασία πρέπει να διατηρεί ένα δείκτη σε ένα μονοπάτι που οδηγεί στη ρίζα, καθώς και μία ουρά όπου βρίσκονται οι αιτήσεις της ίδιας ή και άλλων διεργασιών που δεν έχουν λάβει ακόμα τη σκυτάλη.

Αμοιβαίος Αποκλεισμός: Αλγόριθμος Raymond



Αμοιβαίος Αποκλεισμός: Αλγόριθμος Raymond



Δρομολόγηση

- Η διαδικασία με την οποία ένας κόμβος επιλέγει έναν ή περισσότερους γειτονικούς κόμβους για να προωθήσει ένα πακέτο πληροφορίας προς τον τελικό του στόχο
- Πίνακες δρομολόγησης: δομές που κρατούν πληροφορίες σε κάθε κόμβο για την τοπολογία του δικτύου

Κριτήρια απόδοσης αλγορίθμων δρομολόγησης

- Ορθότητα
- Πολυπλοκότητα
- Αποδοτικότητα
- Σταθερότητα
- Προσαρμοστικότητα
- Δικαιοσύνη

Αλγόριθμος Chandy - Misra

- Υπολογίζει όλα τα ελάχιστα μονοπάτια προς ένα προορισμό, χρησιμοποιώντας ένα κατανεμημένο υπολογισμό που αρχικοποιείται από ένα απλό κόμβο και στον οποίο συμμετέχουν και άλλοι κόμβοι μόνο αφού λάβουν κάποιο μήνυμα.
- Θεωρούμε ότι κάθε κόμβος γνωρίζει ποιοι είναι οι γείτονές του και το κόστος της σύνδεσης με καθένα από αυτούς.
- Για τον υπολογισμό, για όλους τους κόμβους, της απόστασης προς τον κόμβο u_0 (και ενός προτεινόμενου καναλιού), κάθε κόμβος u ξεκινά αρχικοποιώντας την απόσταση $D_u[u_0]$ ίση με άπειρο και περιμένει για την παραλαβή μηνυμάτων

Αλγόριθμος Chandy - Misra

- Ο κόμβος u_0 στέλνει ένα μήνυμα <mydist, u_0,0>, όπου δηλώνει την απόσταση από τον εαυτό του ξεκινώντας τον αλγόριθμο.
- Όταν ένας κόμβος u λαμβάνει ένα μήνυμα <mydist, u_0,d> από τον γείτονα w, όπου η απόσταση $d+w_{uw} < D_u[u_0]$, ο u θέτει $D_u[u_0]$ ίσο με $d+w_{uw}$ και στέλνει ένα μήνυμα <mydist, u_0, $D_u[u_0]$> προς όλους τους γείτονές του.
- Εάν τα κόστη όλων των καναλιών θεωρούνται ότι είναι ίσα, όλα τα συντομότερα μονοπάτια προς τον κόμβο u_0 υπολογίζονται ανταλλάσσοντας $O(N^2)$ μηνύματα ανά κανάλι και $O(N^2*|E|)$ μηνύματα συνολικά.

Αλγόριθμος Chandy - Misra

Var $D_u[u_0]$: weight, init ∞; $Nb_u[u_0]$:node, init undefined;

For node u0 only:

Begin

$Du_0[u_0]=0$;

Forall $w \in Neigh_{u0}$ do send <mydist, u_0, 0> to w

End

Processing a <mydist, u_0, d> message from neighbour w by u:

Begin

Receive <mydist, u_0, d> from w;

If $d+w_{uw} < D_u[u_0]$ then {

$D_u[u_0]= d+w_{uw}$; $Nb_u[u_0]=w$;

Forall $x \in Neigh_u$ do send <mydist, u_0, $D_u[u_0]$> to x

}

End

Δημιουργία πινάκων δρομολόγησης – Αλγόριθμος Floyd – Warshall

- Είναι μη κατανεμημένος αλγόριθμος
- Σε κάθε link της τοπολογίας αντιστοιχίζεται ένα βάρος
- Υπολογίζει το μονοπάτι με το μικρότερο βάρος μεταξύ δύο οποιωνδήποτε κόμβων μιας τοπολογίας δικτύου
- Αρχικά υπολογίζει το βάρος κάθε σύνδεσης με γειτονικούς κόμβους (απόσταση 1) και μετά επεκτείνεται για όλους τους υπόλοιπους κόμβους μέχρι να τους καλύψει όλους (απόσταση ≥2)

Αλγόριθμος Floyd – Warshall - Πολυπλοκότητα

- Η βασική επανάληψη του αλγορίθμου εκτελείται Ν φορές και περιλαμβάνει N^2 λειτουργίες (οι οποίες μπορούν να εκτελεστούν παράλληλα ή σειριακά)
- Άρα η απόσταση ανάμεσα σε δύο οποιουσδήποτε κόμβους εκτελείται σε $\Theta(N^3)$ βήματα.

Αλγόριθμος του Toueg (απλός αλγόριθμος)

- Είναι κατανεμημένος αλγόριθμος
- Βασίζεται στον αλγόριθμο των Floyd- Warshall
- Κάθε κόμβος γνωρίζει τους γείτονές του και το βάρος κάθε γειτονικής σύνδεσης
- Ο αλγόριθμος ανταλλάσσει O(N) μηνύματα ανά κανάλι και O(N*|E|) συνολικά.
- Κάθε κόμβος u εκτελεί τον εξής αλγόριθμο:

Begin S_u:=Ø;

forall v ∈ V do

if v=u then begin $D_u[v]:=0$; $Nb_u[v]:=udef$ end

Αλγόριθμος του Toueg (απλός αλγόριθμος) (συνέχεια)

```
    else if v ∈ Neigh_u
            then begin D_u[v]:=w_uv; Nb_u[v]:=v end
    else begin begin D_u[v]:=∞; Nb_u[v]:=udef end;
  while S_u ≠ V do
    begin pick w from V\ S_u /* the same node */
            if u=w then broadcast the table D_w
                   else receive the table D_w
            forall v ∈ V do
                   if D_u[w]+ D_w[v]< D_u[v] then
                          begin D_u[v]:= D_u[w]+ D_w[v];
                                 Nb_u[v]:=Nb_u[w]
                          end;
            S_u:=S_u ∪ {w}
    end
end
```

Ο αλγόριθμος Merlin-Segall

- Ορισμός: Για κάθε d ϵ V υπάρχει ένα δένδρο T_d=(V, Ed) τέτοιο ώστε $E_d \subseteq E$ ώστε για κάθε κόμβο v ϵ V, το μονοπάτι από το v στο d μέσα από το δένδρο T_d είναι ένα βέλτιστο μονοπάτι από το v στο d γράφο G.
- Για κάθε προορισμό v, κάθε κόμβος u διατηρεί μια εκτίμηση για την απόσταση από το v ($D_u[v]$) και για τον γείτονα κόμβο προς τον οποίο προωθούνται πακέτα προς τον u ($Nb_u[v]$), ο οποίος είναι και ο πατέρας του u στο T_v.
- Σε κάθε γύρο ενημέρωσης κάθε κόμβος u στέλνει την εκτιμώμενη απόστασή του $D_u[v]$, σε όλους τους γείτονές του , εκτός από τον $Nb_u[v]$
- Εάν ο κόμβος u λάβει από ένα γείτονα w ένα μήνυμα <mydist, v,d> και αν $d+w_{vw} < D_u[v]$, ο u θέτει $Nb_u[v]=w$ και $D_u[v]= d+w_{vw}$

Ο αλγόριθμος Merlin-Segall

- Ο γύρος ενημέρωσης ελέγχεται από τον v και απαιτεί την ανταλλαγή 2 μηνυμάτων από κάθε κανάλι.
- Μετά από i γύρους ενημέρωσης θα έχουν υπολογιστεί όλα τα κοντινότερα μονοπάτια σε το πολύ i βήματα, ώστε μετά από N γύρους θα έχουν υπολογιστεί όλα τα κοντινότερα μονοπάτια προς τον v.
- Σε κάθε γύρο ενημέρωσης κάθε κόμβος u στέλνει την εκτιμώμενη απόστασή του $D_u[v]$, σε όλους τους γείτονές του , εκτός από τον $Nb_u[v]$
- Ανταλλάσσονται τελικά $O(N^2)$ μηνύματα ανά κανάλι και $O(N^2*|E|)$ συνολικά.

Έλεγχος τερματισμού (termination detection)

- Ο υπολογισμός ενός κατανεμημένου αλγορίθμου τερματίζει όταν ο αλγόριθμος φτάνει σε μια τερματική διαμόρφωση
- Δεν συμβαίνει πάντα σε μια τερματική διαμόρφωση του αλγορίθμου κάθε διεργασία να είναι σε τερματική κατάσταση
- Με ποιο τρόπο μπορούν οι διεργασίες να μάθουν τον τερματισμό ενός αλγορίθμου;

Κανόνες βασικού αλγορίθμου

- Κάθε διεργασία μπορεί να βρίσκεται σε μία από τις καταστάσεις: ενεργή (active), παθητική (passive) και internal
- Μια διεργασία γίνεται ενεργή όταν συμβαίνει ένα εσωτερικό γεγονός ή ένα γεγονός αποστολής μηνύματος
- Μια διεργασία γίνεται παθητική μόνο μετά από κάποιο εσωτερικό γεγονός
- Μια διεργασία γίνεται ενεργή αφού λάβει ένα μήνυμα.

Αλγόριθμος Dijkstra

- Ο σκοπός είναι να δώσει τη δυνατότητα σε έναν από τους κόμβους (στο συντονιστή) να ανακαλύψει αν έχει επιτευχθεί μια σταθερή κατάσταση, δηλαδή μια κατάσταση κατά την οποία όλοι οι κόμβοι είναι παθητικοί και κανένα μήνυμα δεν βρίσκεται σε φάση μετάδοσης.
- Αναφέρεται σε δίκτυα με τοπολογία δακτυλίου.
- Κάθε κόμβος διατηρεί ένα μετρητή c, αρχικά ίσο με 0.
- Η αποστολή ενός μηνύματος αυξάνει τον c κατά 1. Η λήψη ενός μηνύματος μειώνει τον c κατά 1.
- Άρα το άθροισμα όλων των μετρητών c ισούται με τον αριθμό των μηνυμάτων που η παράδοσή τους εκκρεμεί ακόμη. Στο δακτύλιο κινείται μία σκυτάλη, η οποία όπως και κάθε κόμβος έχει ένα χρώμα.

Αλγόριθμος Dijkstra

- Αρχικά όλοι οι κόμβοι και η σκυτάλη έχουν χρώμα λευκό.
- Όταν ένας κόμβος λαμβάνει ένα μήνυμα του βασικού αλγορίθμου (όχι τη σκυτάλη δηλαδή), γίνεται μαύρος.
- Όταν ένας κόμβος προωθεί τη σκυτάλη γίνεται λευκός.
- Όταν ένας μαύρος κόμβος προωθεί τη σκυτάλη, τότε η σκυτάλη γίνεται μαύρη. Διαφορετικά η σκυτάλη διατηρεί το χρώμα της.

Αλγόριθμος Dijkstra

Ο κόμβος 0 αρχικοποιεί τον αλγόριθμο στέλνοντας μία σκυτάλη με την τιμή 0 στον κόμβο N-1

Όταν ο κόμβος i (i≠0) λάβει τη σκυτάλη

Κρατάει τη σκυτάλη μέχρι ο κόμβος να γίνει παθητικός

Στέλνει τη σκυτάλη στον κόμβο i-1 αυξάνοντας την τιμή της σκυτάλης κατά c

Αν ο κόμβος είναι μαύρος η σκυτάλη γίνεται μαύρη

Ο κόμβος γίνεται λευκός

Όταν ένας κόμβος λάβει ένα μήνυμα του βασικού αλγορίθμου

Γίνεται μαύρος

Όταν ο κόμβος 0 λάβει τη σκυτάλη

Αν είναι παθητικός και λευκός

Αν η σκυτάλη είναι λευκή

Αν το άθροισμα της τιμής της σκυτάλης και του c είναι 0

Έχουμε τερματισμό

Αλλιώς

Ο κόμβος 0 ξεκινά νέο γύρο στον αλγόριθμο

Ο αλγόριθμος του Mattern

- Ανιχνεύει τερματισμό πολύ γρήγορα, για την ακρίβεια, μία χρονική μονάδα μετά τη χρονική στιγμή που συνέβη.
- Ο αλγόριθμος ανιχνεύει τον τερματισμό ενός κεντρικοποιημένου υπολογισμού
- Θεωρεί ότι κάθε διεργασία μπορεί να στείλει ένα μήνυμα στον αρχικοποιητή του υπολογισμού απ' ευθείας.
- Σε κάθε μήνυμα και κάθε διεργασία ανατίθεται μια τιμή πιστώσεως (credit value) η οποία είναι συνεχώς μεταξύ 0 και 1

Ο αλγόριθμος του Mattern

- Διατηρεί τις ακόλουθες θεωρήσεις σαν αμετάβλητες συνθήκες:
  - Το άθροισμα όλων των πιστώσεων (μηνυμάτων και διεργασιών) είναι 1.
  - Ένα μήνυμα του βασικού αλγορίθμου έχει θετική τιμή πίστωσης.
  - Μία ενεργή διεργασία έχει θετική τιμή πίστωσης.
- Μία διεργασία που κατέχει μια θετική τιμή πίστωσης αλλά δεν πληροί κάποια από τις παραπάνω τρεις συνθήκες (π.χ. μια παθητική διεργασία) στέλνει την τιμή της πίστωσής της στον αρχικοποιητή.
- Ο αρχικοποιητής λειτουργεί σαν τράπεζα, συλλέγοντας όλες τις πιστώσεις που στέλνονται σε αυτόν σε μία μεταβλητή που καλείται ret.
- Όταν ο αρχικοποιητής κατέχει όλες τις πιστώσεις, σημαίνει ότι δεν υπάρχουν τέτοιες διεργασίες και τέτοια μηνύματα στο δίκτυο. Άρα υπάρχει τερματισμός.
- Έτσι όταν ret=1 ο αρχικοποιητής ανακοινώνει τον τερματισμό

Ο αλγόριθμος του Mattern

- Όταν μία ενεργή διεργασία στέλνει ένα μήνυμα, η πίστωσή της μοιράζεται ανάμεσα σε αυτή και το μήνυμα.
- Όταν μία διεργασία καθίσταται ενεργή παίρνει σαν τιμή πίστωσης αυτή που φέρει το μήνυμα που την ενεργοποιεί.
- Στην περίπτωση που μία διεργασία λάβει ένα μήνυμα ενώ είναι ήδη ενεργή, δεν χρειάζεται την πίστωση που αυτό φέρει, αλλά ούτε και την καταστρέφει.
- Η διεργασία τότε κάνει ένα από τα δύο παρακάτω, που και τα δύο οδηγούν σε σωστό αλγόριθμο:
  - Η πίστωση που φέρει το μήνυμα στέλνεται στον αρχικοποιητή.
  - Η πίστωση που φέρει το μήνυμα προστίθεται στην πίστωση της διεργασίας.

Ο αλγόριθμος του Mattern

var $state_p$: (active,passive) init if $p=p_0$ then active else passive;

$cred_p$: [0..1] init if $p=p_0$ then 1 else 0;

ret : [0..1] init 0; /* for p_0 only */

S_p: /* state=active */

send<mes,$cred_p$/2>; $cred_p$:=$cred_p$/2;

R_p: /* A message <mes,c> has arrived at p */

receive <mes,c>; $state_p$:=active; $cred_p$:=$cred_p$+c;

Ip: /* $state_p$=active */

$state_p$:=passive; send<ret,$cred_p$> to p_0; $cred_p$:=0;

A_{p0}: /* A <ret,c> message has arrived at p_0 */

receive <ret,c>; ret:=ret+c; if ret=1 then Announce_termination;

Ο αλγόριθμος των Dijkstra - Scholten

- Ο αλγόριθμος τερματισμού διατηρεί ένα κατευθυνόμενο δένδρο με τις ακόλουθες ιδιότητες:
  - Το δένδρο είναι άδειο ή έχει ρίζα τον αρχικοποιητή
  - Περιλαμβάνει όλες τις ενεργές διεργασίες του δικτύου και όλα τα μηνύματα του βασικού αλγορίθμου που μεταδίδονται
- Ο αλγόριθμος σβήνει επίσης κόμβους από το δένδρο:
  - Όταν ένα μήνυμα παραλαμβάνεται
  - Όταν μια διεργασία γίνεται παθητική και δεν έχει εκκρεμή μηνύματα
- Όταν ένα μήνυμα παραληφθεί ή μια διεργασία δεν έχει «παιδιά» ειδοποιεί με κατάλληλο μήνυμα τον «πατέρα» της στο δένδρο

Ο αλγόριθμος των Dijkstra - Scholten

var $state_p$: (active, passive) init if $p=p_0$ then active else passive; /* p_0 είναι ο αρχικοποιητής του βασικού αλγορίθμου */

sc_p : integer init 0;

$father_p$: P init if $p=p_0$ then p else udef;

S_p : {$state_p$=active} /* αποστολή μηνύματος με την ταυτότητα του κόμβου */

begin send <mes,p>; sc_p:=sc_p+1 end

R_p: { A message <mes,q> has arrived at p }

begin receive <mes,q>;

$state_p$:=active;

if $father_p$=udef then $father_p$:=q

else send <sig,q> to q

end

Ο αλγόριθμος των Dijkstra - Scholten

I_p: {state=active}
begin
$state_p$:=passive;
if sc_p=0 then {
if $father_p$=p then Announce
else send <sig, $father_p$> to $father_p$;
$father_p$:=udef
}
end
A_p: {A signal <sig, p> arrives at p}
begin
receive <sig,p>;
sc_p:=sc_p-1;
if sc_p=0 and state=passive then {
if $father_p$=p then Announce
else send <sig, $father_p$> to $father_p$;
$father_p$:=udef;
}
end

Ανοχή σε σφάλματα

- Θεωρούμε ότι μπορεί να υπάρξουν σφάλματα σε διεργασίες.
- Υπάρχουν δύο είδη αλγορίθμων:
  - Αλγόριθμοι ευρωστίας (robust algorithms): κάθε βήμα μιας διεργασίας γίνεται με προσοχή για να επιβεβαιώσει ότι παρά τα όποια σφάλματα, οι σωστές διεργασίες εκτελούν σωστά βήματα.
  - Αλγόριθμοι σταθεροποίησης: οι σωστές διεργασίες μπορεί να επηρεαστούν από σφάλματα, αλλά ο αλγόριθμος εγγυάται την επανάκαμψη από οποιαδήποτε διαμόρφωση όταν η διεργασία επανέλθει σε σωστή συμπεριφορά.

Μοντέλα σφαλμάτων

- Αρχικά-νεκρές διεργασίες: μία διεργασία δεν εκτελεί κανένα βήμα του τοπικού της αλγορίθμου
- Μοντέλο κατάρρευσης: μια διεργασία θεωρείται ότι καταρρέει εάν εκτελεί τον τοπικό της αλγόριθμο σωστά μέχρι ενός χρονικού σημείου, και δεν εκτελεί κανένα βήμα μετά.
- Βυζαντινή συμπεριφορά (byzantine behavior): μία διεργασία καλείται βυζαντινή εάν εκτελεί βήματα τα οποία είναι αυθαίρετα βήματα και όχι σύμφωνα με τον τοπικό της αλγόριθμο. Συγκεκριμένα, στέλνει μηνύματα με αυθαίρετο περιεχόμενο.

Αλγόριθμοι ευρωστίας έναντι αλγορίθμων σταθεροποίησης

- Οι αλγόριθμοι σταθεροποίησης προσφέρουν προστασία από προσωρινά σφάλματα, π.χ. προσωρινή ανώμαλη συμπεριφορά συστατικών του συστήματος.
  - Αυτά τα σφάλματα μπορεί να συμβούν σε μεγάλα τμήματα ενός κατανεμημένου συστήματος όταν οι φυσικές συνθήκες προσωρινά φτάσουν ακραίες τιμές, εισάγοντας εσφαλμένη συμπεριφορά μνημών και επεξεργαστών.
- Οι αλγόριθμοι ευρωστίας ασχολούνται με τα μόνιμα σφάλματα από ένα περιορισμένο σύνολο από συστατικά στοιχεία του συστήματος. Οι υπόλοιπες διεργασίες δεν επηρεάζονται από αυτά τα σφάλματα και παραμένουν σωστές.

Αλγόριθμος συμφωνίας σε ένα υποσύνολο σωστών διεργασιών

- Αναφέρεται στο μοντέλο αρχικά-νεκρών διεργασιών
- Είναι γνωστός ως αλγόριθμος των Fischer, Lynch & Paterson.
- Κάθε σωστή διεργασία υπολογίζει το ίδιο σύνολο σωστών διεργασιών.
- Έστω ότι υπάρχουν τουλάχιστον L σωστές διεργασίες.
- Ο αλγόριθμος εκτελείται σε δύο φάσεις.
- Οι διεργασίες κατασκευάζουν ένα κατευθυνόμενο γράφο G ανακοινώνοντας την ταυτότητά τους και περιμένουν για την λήψη L μηνυμάτων.

Αλγόριθμος συμφωνίας σε ένα υποσύνολο σωστών διεργασιών 2

- Καθώς υπάρχουν τουλάχιστον L σωστές διεργασίες, κάθε σωστή διεργασία λαμβάνει καταλλήλως πολλά μηνύματα για να ολοκληρώσει αυτό το έργο.
- Οι επόμενοι της p σε ένα γράφο G είναι οι κόμβοι q από τους οποίους η p έχει λάβει ένα μήνυμα <name,q>.
- Μία αρχικά-νεκρή διεργασία δεν στέλνει ούτε λαμβάνει μηνύματα σχηματίζοντας έτσι ένα απομονωμένο κόμβο στον G.
- Ένας «κόμπος» (knot) είναι ένα ισχυρά συνδεδεμένο συστατικό μέρος του γράφου, χωρίς εξερχόμενες ακμές, που περιέχει τουλάχιστον δύο κορυφές.

Αλγόριθμος συμφωνίας σε ένα υποσύνολο σωστών διεργασιών 3

- Στη δεύτερη φάση οι διεργασίες κατασκευάζουν ένα παραγόμενο υπογράφο του G, που περιέχει τουλάχιστον τους απογόνους τους στο κατευθυνόμενο δένδρο, λαμβάνοντας το σύνολο των επόμενων κάθε διεργασίας που γνωρίζουν ότι είναι σωστή.
- Στο τέλος του αλγορίθμου κάθε σωστή διεργασία έχει λάβει το σύνολο των επόμενων καθεμιάς από τους απογόνους της, πράγμα που της επιτρέπει να υπολογίσει τον μοναδικό «κόμπο» στον G γράφο.

Σταθεροποίηση (stabilization)

- Οι σταθεροποιούμενοι αλγόριθμοι ακολουθούν μια πολιτική η οποία να προκαλέσει σε μια διεργασία να συμπεριφέρεται με αστάθεια, αλλά εγγυάται την επιστροφή σε σωστή συμπεριφορά σε περιορισμένο αριθμό βημάτων
- Οι διεργασίες θεωρείται ότι δουλεύουν σωστά, αλλά η συνολική διαμόρφωση του συστήματος μπορεί να διαταραχθεί από κάποια βλάβη.
- Ένας αλγόριθμος θεωρούμε ότι είναι αυτό-σταθεροποιούμενος αν μπορεί να ξεκινήσει να συμπεριφέρεται σωστά ανεξάρτητα από τη διαμόρφωση του συστήματος.

Ιδιότητες σταθεροποιούμενων αλγορίθμων

- Ανοχή σε λάθη: επανέρχεται από οποιαδήποτε διαμόρφωση
- Αρχικοποίηση: δεν χρειάζεται κάποια ειδική αρχικοποίηση
- Δυναμική τοπολογία: Συγκλίνει σε λύση ακόμα και μετά από αλλαγή στην τοπολογία
- Αρχικές ασυμβατότητες: μπορεί αρχικά να έχει ασταθή έξοδο
- Υψηλή πολυπλοκότητα
- Δεν ανακαλύπτεται η σταθεροποίηση

Ο αλγόριθμος token ring του Dijkstra

- Είναι ο πρώτος σταθεροποιούμενος αλγόριθμος (1974)
- Αφορά αμοιβαίο αποκλεισμό σε ένα δακτύλιο διεργασιών
- Οι διεργασίες εκτελούν κώδικα σε κρίσιμες περιοχές αλλά μόνο μία διεργασία μπορεί να εκτελεί τον κώδικα αυτό κάθε χρονική στιγμή.
- Κάθε διεργασία πρέπει να αποκτά το δικαίωμα εκτέλεσης του κώδικα συχνά.
- Η λύση δίνεται με το να γυρνά ένα token στο δακτύλιο και η διεργασία που το κατέχει εκτελεί τον κώδικα στην κρίσιμη περιοχή

Ο αλγόριθμος token ring του Dijkstra (περιγραφή λύσης)

- Κάθε διεργασία έχει μια κατάσταση που δηλώνεται με ένα ακέραιο που ξεπερνά το μέγεθος του δακτυλίου
- Κάθε διεργασία μπορεί να διαβάσει την κατάσταση τη δική της και της προηγούμενής της στο δακτύλιο
- Μια διεργασία $i \neq 0$ μπαίνει στην κρίσιμη περιοχή όταν η κατάστασή της είναι διαφορετική από αυτή της προηγούμενής της
- Η διεργασία $i=0$ μπαίνει στην κρίσιμη περιοχή όταν η κατάστασή της είναι ίδια με αυτή της προηγούμενής της $(N-1)$

Προσανατολισμός δακτυλίου

- Θεωρούμε ένα μη κατευθυνόμενο δακτύλιο από N διεργασίες
- Κάθε διεργασία έχει ονομάσει τη μία σύνδεσή της «προηγούμενη» και την άλλη «επόμενη»
- Ο σύνδεσμος l για τη σύνδεση των διεργασιών p,q καλείται l_{pq}
- Ο αλγόριθμος πρέπει να ικανοποιεί τη συνθήκη: για κάθε διεργασία p το l_{pq} είναι «επόμενη» αν και μόνο αν το l_{qp} είναι «προηγούμενη»
- Κάθε διεργασία βρίσκεται σε μία από τις καταστάσεις S (send), R (receive), I (internal)
- Ιδέα: Οι διεργασίες κυκλοφορούν κυκλικά στο δακτύλιο tokens, και κάθε διεργασία θέτει σαν επόμενό της την κατεύθυνση του τελευταίου token που έστειλε. Αν δυο token συναντηθούν το ένα εξαφανίζεται

Προσανατολισμός δακτυλίου

- Τελική κατάσταση: όλα τα εναπομείναντα tokens ταξιδεύουν προς την ίδια κατεύθυνση στο δακτύλιο (προσανατολισμένος δακτύλιος)
- $State_p$=I και $state_q$=S και l_{qp}=«επόμενη» τότε $state_p$=R; if l_{pq}=«επόμενη» then flip
- $State_p$=S και $state_q$=R και l_{qp}=«προηγούμενη» και l_{pq}=«επόμενη» τότε $state_p$=I
- $State_p$=R και l_{pq}=«προηγούμενη» και !($state_q$=S και l_{qp}=«επόμενη») τότε $state_p$=S
- $State_p$= $state_q$=S και l_{qp} =l_{pq} =«επόμενη» τότε $state_p$=R; flip
- $State_p$= $state_q$=I και l_{qp} =l_{pq} =«επόμενη» τότε $state_p$=S