6η σειρά ασκήσεων

Άλκης Γεωργόπουλος Α.Μ. 39
Αναστάσιος Κοντογιώργης Α.Μ. 43

Απαντήσεις

Άσκηση 1.

Απάντηση

Η αλλαγή ενός ρολογιού προς τα πίσω μπορεί να προκαλέσει ανεπιθύμητη συμπεριφορά σε κάποια προγράμματα. Για παράδειγμα, έστω ότι ένας χρήστης αποθηκεύει το αρχείο mylib.c την χρονική στιγμή 10:27:53.0. Μετά από 3 δευτερόλεπτα το ρολόι έχει γυρίσει προς τα πίσω και δείχνει 10:27:51.0. Κάποιος άλλος χρήστης θέλει να μεταγλωττίσει το αρχείο myprog.c το οποίο χρησιμοποιεί το mylib.c, οπότε καλεί το πρόγραμμα make. Όμως το make βλέπει ότι το mylib.o έχει timestamp 10:27:53.0 ενώ το mylib.c έχει 10:27:51.0. Έτσι δεν μεταγλωττίζει το mylib.c, χρησιμοποιεί την παλιά έκδοση του mylib.o και τελικά το εκτελέσιμο πρόγραμμα μπορεί να παρουσιάζει προβλήματα λόγω ασυμβατότητας εκδόσεων (π.χ. διαφορετικές παράμετροι σε συναρτήσεις) ενώ ο πηγαίος κώδικας να είναι σωστός.

Για να διορθωθεί η απόκλιση των 4 δευτερολέπτων μέσα σε 8 δευτερόλεπτα, αρκεί το ρολόι να τρέχει δυο φορές πιο αργά για 8 δευτερόλεπτα. Δηλαδή αν ο timer προκαλεί 100 interrupts ανά δευτερόλεπτο και ο software counter αυξάνεται κάθε φορά κατά 10, τώρα θα πρέπει να αυξάνεται κατά 5. Για παράδειγμα, αν ο counter την χρονική στιγμή 10:27:54.0 έχει τιμή 37674000, στο επόμενο interrupt θα πρέπει να του ανατεθεί τιμή 37674005 και άρα η ώρα θα είναι ακόμα 10:27:54.0 (το 0,5 δεν θα φαίνεται σε μια απλή κλήση gettime). Στον επόμενο χτύπο ο counter θα γίνει 37674010 και άρα η ώρα θα είναι 10:27:54.1. Συνεχίζοντας με αυτόν τον τρόπο μετά από 8 δευτερόλεπτα το ρολόι θα δείχνει 10:27:58.0 και θα είναι συγχρονισμένο.

Άσκηση 2.

Απάντηση

Σύμφωνα με τον αλγόριθμο του Cristian θα πρέπει να θέσει το ρολόι του στο άθροισμα του timestamp και του χρόνου διάδοσης. Για τον υπολογισμό του χρόνου διάδοσης, όταν ο χρόνος επεξεργασίας δεν είναι γνωστός, βασιζόμαστε μόνο στο round-trip delay, το οποίο διαιρούμε με το 2.

- Αν επιλέξουμε να πάρουμε τον **μέσο όρο** των μετρήσεων, τότε βρίσκουμε ότι η καθυστέρηση διάδοσης t_p είναι (22 + 25 + 20) / 6 = 11.17 msec.
- Αν επιλέξουμε την **μικρότερη καθυστέρηση** σαν την πιο αντιπροσωπευτική, τότε βρίσκουμε ότι η καθυστέρηση διάδοσης t_p είναι 20 / 2 = 10 msec.
- Αν επιλέξουμε να **αγνοήσουμε τις μετρήσεις** πάνω από κάποιο κατώφλι καθυστέρησης, τότε πιθανώς να βρούμε διαφορετικό αποτέλεσμα. Για παράδειγμα αν έχουμε κατώφλι = 23 msec τότε η δεύτερη μέτρηση απορρίπτεται και ο μέσος όρος των άλλων δύο μας δίνει t_p = 11 msec.

Όσον αφορά το **ποιον από αυτούς τους χρόνους** θα πρέπει να χρησιμοποιήσει ο client για το συγχρονισμό του ρολογιού του, το βιβλίο απλά αναφέρει «when the reply comes in, the value in the message can be increased by this amount $[(T_1 - T_2) / 2]$ to give an estimate of the server's current time». Επομένως θα χρησιμοποιήσει απλά τον τελευταίο από τους χρόνους. Βέβαια μπορούν κι εδώ να ακολουθηθούν οι προηγούμενες τεχνικές (μέσος όρος, μικρότερη καθυστέρηση, αγνόηση μετρήσεων πάνω από κατώφλι) οπότε σε κάθε περίπτωση ο client θα χρησιμοποιήσει διαφορετικό χρόνο.

Τελικά αν υποθέσουμε ότι χρησιμοποιεί τον τελευταίο χρόνο και του προσθέτει t_p = 11.17 msec, τότε θα πρέπει να θέσει το ρολόι του στις **10:54:28.353**.

Η πληροφορία ότι ο χρόνος από την αποστολή μέχρι τη λήψη ενός μηνύματος είναι τουλάχιστον 8msec δεν είναι χρήσιμη. Αν δινόταν ότι ο **χρόνος επεξεργασίας** της αίτησης στον server είναι τουλάχιστον 8 msec, τότε θα αφαιρούσαμε 8 msec από όλα τα round – trips, ώστε τελικά σαν round – trips να θεωρούσαμε τα

Round-trip (ms)	**Time (hr:min:sec)**
14	**10:54:23.674**
17	**10:54:25.450**
12	**10:54:28.342**

Στη συνέχεια θα εφαρμόζαμε πάλι οποιαδήποτε από τις παραπάνω παραλλαγές του αλγορίθμου του Cristian ώστε να πάρουμε τα τελικά αποτελέσματα. Για παράδειγμα, αν υποθέσουμε ότι βρίσκει τον μέσο όρο t_p = 7.17 msec και τον προσθέτει στον τελευταίο χρόνο, τότε θα πρέπει να θέσει το ρολόι του στις **10:54:28.349**

Άσκηση 3.

Απάντηση

Διακρίνουμε δύο περιπτώσεις:
1) Εάν ο αριθμός των γεγονότων μεταξύ των e και e' είναι 0, τότε προφανώς ισχύει L(e) = L(e').

2) Θα δείξουμε με επαγωγή ότι ισχύει $e \rightarrow e' \Rightarrow L(e) < L(e')$.
Έστω m ο αριθμός των μηνυμάτων που μεσολαβούν των γεγονότων e και e'.

Για m = 1, έχουμε L(m) = L(e) + 1 και L(e') = L(m) + 1. Άρα L(e') = L(e) +2 > L(e).

Έστω ότι ισχύει για m = k. Δηλαδή όταν μεσολαβούν k γεγονότα δεχόμαστε ότι ισχύει L(e) < L(e').
Θα δείξουμε ότι ισχύει για m = k + 1.
Αν για m = k γεγονότα L(e') = A > L(e) για m = k+1 L(e') = A+1 > L(e).

Οπότε δείξαμε ότι σε κάθε περίπτωση ισχύει $e \rightarrow e' \Rightarrow L(e) \leq L(e')$.

Άσκηση 4.

Απάντηση

(α)

Κατά την ανταλλαγή μηνυμάτων μεταξύ δύο διεργασιών i και j, για κάθε μήνυμα που λαμβάνει η διεργασία i από την διεργασία j, αυξάνει το $V_i[i]$ κατά 1, διατηρώντας το πάντα μεγαλύτερο ή ίσο από το $V_j[j]$.
Υπάρχει όμως η περίπτωση δύο μηνύματα της V_j να φτάσουν σε αντίστροφη σειρά. Τότε πιθανώς $V_i[i] < V_j[j]$, αλλά σ' αυτήν την περίπτωση η διεργασία i θα αναβάλλει το μήνυμα της διεργασίας j που έφτασε πρώτο επειδή δεν είναι στη σειρά που το περιμένει. Έτσι θα επεξεργαστεί πρώτο το "παλιότερο" μήνυμα με αποτέλεσμα πάντα να είναι $V_i[i] \geq V_i[j]$,

(β)

Έστω ότι το γεγονός e έγινε στην διεργασία i και το e' στην j. Από την παραπάνω ανισότητα συμπεραίνουμε ότι $V_i[i] \geq V_i[j]$ και $V_j[j] \geq V_j[i]$. Επειδή όμως η διεργασία i δεν έχει ενημερωθεί για το συμβάν e', είναι $V_i[j] < V_j[j]$ και αντίστοιχα για την διεργασία j είναι Vj[i] < Vi[i]. Επομένως δεν μπορεί να ισχύει ούτε $V(e) \leq V(e')$ αλλά ούτε $V(e') \leq V(e)$.
Τελικά αν ισχύει V(e) < V(e') τότε με απαγωγή σε άτοπο μπορούμε να συμπεράνουμε ότι $e \rightarrow e'$, γιατί αν ήταν παράλληλα τότε δείξαμε ότι η προηγούμενη ανισότητα δεν μπορεί να ισχύει, ενώ αν ίσχυε ότι $e' \rightarrow e$ τότε εξ' ορισμού θα ήταν V(e) > V(e'), δηλαδή άτοπο.

Άσκηση 5.

Απάντηση

Με την χρήση των timestamps του Lamport, δύο παράλληλα γεγονότα (π.χ. δύο updates) παραμένουν ασυσχέτιστα (δεν υπάρχει σχέση a ? b ώστε να διαταχθούν χρονικά). Έτσι είναι αδύνατο να επιτευχθεί totally ordered multicasting.

Όμως όταν κάθε κόμβος στέλνει acknowledgement, τότε το acknowledgement του a σαφώς έπεται του γεγονότος a, αλλά μπορεί επίσης να διαταχθεί χρονικά και σε σχέση με το b, γιατί θα του έχει βάλει timestamp η διεργασία που προκάλεσε το b. Έτσι με την χρήση acknowledgements μπορούμε να διατάξουμε χρονικά τα ίδια τα acknowledgements και να πετύχουμε totally ordered multicasting, θεωρώντας ότι τα a και b έγιναν με την ίδια σειρά όπως και τα acknowledgements τους.

Άσκηση 6.

Απάντηση

Ο προτεινόμενος αλγόριθμος λύνει το πρόβλημα των περισσοτέρων από ένα ELECTION μηνυμάτων σε ένα δακτύλιο. Για λόγους απλότητας στη περιγραφή θα θεωρήσουμε ότι υπάρχουν δύο μηνύματα που διασχίζουν το δακτύλιο και πρέπει να παραμείνει μόνο το ένα. Παρόλα αυτά ο αλγόριθμος που θα περιγράψουμε γενικεύεται για n αριθμό ELECTION μηνυμάτων.
Υποθέτουμε ότι ο Α και ο Β αντιλαμβάνονται ταυτόχρονα ότι ο προηγούμενος coordinator δεν λειτουργεί πλέον και προωθούν ταυτόχρονα στο δακτύλιο δύο ELECTION μηνύματα. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι ο Α (ένας από τους δύο δηλαδή) βρίσκεται πριν από το Β στη διάταξη του δακτυλίου.
Οπότε ο Β μετά από κάποιο αριθμό βημάτων θα πάρει το ELECTION μήνυμα από τον Α.
Ο Β θα δει ότι δεν υπάρχει στη λίστα του μηνύματος το δικό του process id, οπότε θα καταλάβει ότι δεν είναι το μήνυμα που αυτός προώθησε στο δακτύλιο αλλά ένα άλλο μήνυμα από άλλη διεργασία.
Τότε απλά δεν στέλνει το μήνυμα στην επόμενη διεργασία που βρίσκεται μετά από αυτόν, οπότε και το επιπλέον μήνυμα σταματάει την άσκοπη διάσχιση του δακτυλίου.

Άσκηση 7.

Απάντηση

Το ζητούμενο throughput ορίζεται ως ο μέγιστος αριθμός προσπελάσεων κοινών πόρων που μπορούν να πραγματοποιηθούν στη μονάδα του χρόνου.
Για τον ορισμό του maximum throughput θεωρούμε τον ελάχιστο μέσο χρόνο που απαιτείται για να προσπελάσει μια διεργασία ένα διαμοιραζόμενο πόρο. Ουσιαστικά

είναι ο χρόνος που χρειάζεται για να εισέλθει στη κρίσιμη περιοχή, έστω $t_{εισόδου}$, συν το χρόνο που χρειάζεται να μείνει η διεργασία στη κρίσιμη περιοχή, έστω $t_{εργασίας}$.
Άρα max_throughput = 1 / (min_$t_{εισόδου}$ + min_$t_{εργασίας}$) αμοιβαίοι αποκλεισμοί ανά δευτερόλεπτο.
Από τους γνωστούς αλγορίθμους για επίτευξη αμοιβαίου αποκλεισμού, αυτός με τη μικρότερη καθυστέρηση για την είσοδο μιας διεργασίας σε κρίσιμη περιοχή είναι ο Centralized, που απαιτεί 2 μηνύματα. Αν υποθέσουμε ότι το κάθε μήνυμα απαιτεί χρόνο t1 sec και ο μέσος παραμονής μιας διεργασίας στη κρίσιμη περιοχή είναι t2 sec. Η χρονική καθυστέρηση είναι 2*t1 + t2 sec. Άρα το throughput θα είναι 1/(2*t1 + t2) αμοιβαίοι αποκλεισμοί ανά δευτερόλεπτο.

Άσκηση 8.

Απάντηση

Οι τρεις βασικές λειτουργίες που χαρακτηρίζουν το 2PL ισχύουν και για το κατανεμημένο 2PL.
Σε αυτή τη περίπτωση όμως ισχύουν επιπλέον οι εξής θεωρήσεις:

- Τα δεδομένα βρίσκονται κατανεμημένα σε αντίγραφα σε πολλά μηχανήματα του συστήματος.
- Κάθε μηχάνημα διαθέτει τον δικό του scheduler που διαχειρίζεται τις αιτήσεις για transactions
- Επιπλέον διαθέτει και τον δικό του data manager στον οποίο προωθούνται οι αιτήσεις για προσπέλαση στα τοπικά δεδομένα

Σύμφωνα με αυτή την επέκταση του ορισμού, το κατανεμημένο 2PL μπορεί να πραγματοποιήσει κατανεμημένα transactions εκτελώντας τις τρεις βασικές 2PL λειτουργίες, αλλά στα τοπικά δεδομένα κάθε μηχανής.