Η/Υ Ε-07: Κατανεμημένα Συστήματα
Εαρινό Εξάμηνο Ακ. Έτους 2008-2009
Διδάσκουσα: Παναγιώτα Φατούρου
Προγραμματιστικές Εργασίες

Ημερομηνία Παράδοσης:

Σχόλια: Θα πρέπει να στείλετε με e-mail την εργασία σας στη διεύθυνση cse07@cs.uoi.gr μέχρι τις 11:59 της Τετάρτης 3/6/09. Η εξέταση της εργασίας θα γίνει την Παρασκευή, 5/6/09.

Επιπρόσθετα της αποστολής της εργασίας μέσω ηλεκτρονικού ταχυδρομείου, θα πρέπει να αποθηκεύσετε την άσκησή σας σε κάποιο directory στην περιοχή σας χωρίς να αλλάξετε την ημερομηνία της μέχρι να γίνει η εξέτασή της. Η ημερομηνία τελευταίας εγγραφής της άσκησής σας θα πρέπει να είναι είτε κάποια προηγούμενη ή η αναγραφόμενη ημερομηνία και ώρα παράδοσης.

Ο κώδικάς σας θα πρέπει να είναι γραμμένος σε C, να είναι ευανάγνωστος και να έχει σχολιαστεί κατάλληλα.

1ο Πακέτο Εργασιών

Σκοπός

Ο σκοπός των εργασιών αυτών είναι:

- ✓ Η εξοικείωση με τον προγραμματισμό GPUs (Graphical Processor Units), Επεξεργαστικών Μονάδων Καρτών Γραφικών.

Εγχειρίδια προγραμματισμού επεξεργαστικών μονάδων καρτών γραφικών παρέχονται μέσω των ακόλουθων σύνδεσμων:

Τύπου AMD (AMD Stream)

http://developer.amd.com/gpu_assets/Stream_Computing_User_Guide.pdf

Τύπου NVIDIA (NVIDIA CUDA)

http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

Η επεξεργαστική ισχύς που προσφέρουν οι επεξεργαστικές μονάδες καρτών γραφικών αυξάνεται με πολύ γρήγορους ρυθμούς, ενώ παράλληλα υποστηρίζουν ένα πλήρες σύνολο εντολών. Είναι άξιο αναφοράς ότι οι σύγχρονες επεξεργαστικές μονάδες καρτών γραφικών προσφέρουν επεξεργαστική ισχύ πολύ μεγαλύτερη από αυτή των ταχύτερων συμβατικών επεξεργαστών, η οποία είναι της τάξης των Tera-Flops. Τα τελευταία χρόνια αρχίζουν να αναζητούνται τρόποι για την εκμετάλλευση όλης αυτής της επεξεργαστικής ισχύς σε εφαρμογές διαφορετικές από αυτές των γραφικών. Ιδιαίτερο ερευνητικό ενδιαφέρον προκαλεί η ενδεχόμενη αποτελεσματικότητα των επεξεργαστών γραφικών για την υλοποίηση κατανεμημένων δομών δεδομένων. Σε αυτή την εργασία θα πρέπει να μελετήσετε τον

τρόπο προγραμματισμού σύγχρονων επεξεργαστών γραφικών (ATI stream), καθώς και να προγραμματίσετε μερικά κατανεμημένα αντικείμενα.

Εργασία 1

Σε αυτή την εργασία θα υλοποιηθούν αλγόριθμοι ατομικών στιγμιοτύπων κοινόχρηστης μνήμης και θα πραγματοποιηθούν πειραματικές μετρήσεις ώστε να συγκριθεί η αποτελεσματικότητα των αλγορίθμων αυτών. Πιο συγκεκριμένα, θα μελετηθούν και θα υλοποιηθούν οι αλγόριθμοι που παρουσιάζονται στην εργασία:

- Panagiota Fatourou and Nikolaos D. Kallimanis: Time-optimal, space-efficient single-scanner snapshots & multi-scanner snapshots using CAS. PODC 2007: 33-42 (η εργασία είναι διαθέσιμη μέσω του βοηθού του μαθήματος Ν. Καλλιμάνη)
- P. Fatourou and N. Kallimanis, ``Single-Scanner Multi-Writer Snapshot Implementations are Fast!'', *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'06),* pp. 228-237, Denver, Colorado, July 2006.

Εργασία 2

Σε αυτή την εργασία θα υλοποιηθούν κατανεμημένες δομές δεδομένων, όπως ουρές και λίστες. Πιο συγκεκριμένα θα υλοποιηθούν κάποιες από τις κατανεμημένες δομές δεδομένων που παρουσιάζονται στις εργασίες:

- Timothy L. Harris, "A Pragmatic Implementation of Non-blocking Linked-Lists", DISC 2001, LNCS 2180, pp. 300–314, 2001, (http://www.springerlink.com/content/dg5qde5tweprg3gm/fulltext.pdf)
- M. M. Michael and M. L. Scott. `` Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms." *15th ACM Symp. on Principles of Distributed Computing (PODC)*, May 1996. (http://citeseer.ist.psu.edu/michael96simple.html)
- M.M. Michael, "High performance dynamic lock-free hash tables and list-based sets", SPAA 2002. (http://portal.acm.org/citation.cfm?id=564881&dl=GUIDE&coll=GUIDE)

Οι παραπάνω εργασίες εμπεριέχουν τον προγραμματισμό κάποιων κατανεμημένων αλγορίθμων σε επεξεργαστικές μονάδες καρτών γραφικών, την παρουσίαση του μοντέλου αυτού προγραμματισμού στο μάθημα και την προφορική εξέταση του κώδικα που θα υλοποιηθεί.

2ο Πακέτο Εργασιών

Σκοπός

Ο σκοπός των εργασιών αυτών είναι:

✓ Να γνωρίσετε κάποια προγραμματιστικά πακέτα.

- ✓ Να αναπτύξετε μια κατανεμημένη εφαρμογή που ακολουθεί το μοντέλο πελάτη-εξυπηρέτη.

Εργασία 3

Η εφαρμογή που θα πρέπει να υλοποιήσετε είναι ένα σύστημα κράτησης θέσεων για μια αεροπορική εταιρεία. Το σύστημα θα πρέπει να υποστηρίζει ένα συγκεκριμένο αριθμό πτήσεων (MAX_FLIGHTS) και συγκεκριμένο πλήθος διαθέσιμων θέσεων ανά πτήση (για ευκολία υποθέστε ότι ο αριθμός αυτός είναι MAX_SEATS για όλες τις πτήσεις).

Το σύστημα θα πρέπει επίσης να υποστηρίζει τις ακόλουθες λειτουργίες:

1. Κράτηση θέσης (θέσεων) σε μια πτήση της εταιρείας (σαν ορίσματα θα πρέπει να δίνονται ο αριθμός πτήσης και το πλήθος των θέσεων). Μια επιτυχημένη κράτηση θα πρέπει να δίνει ως αποτέλεσμα έναν αύξοντα αριθμό για μελλοντική αναφορά στην κράτηση αυτή, ενώ μια αποτυχημένη κράτηση θα πρέπει να δίνει το κατάλληλο μήνυμα λάθους. Μια πιθανή μορφή της σχετικής συνάρτησης (που καλείται από τον client) θα μπορούσε να είναι η εξής: reservation_no = make_reservation(flight_no, num_seats).
2. Ακύρωση κράτησης θέσης (θέσεων) σε κάποια πτήση της εταιρείας (με όρισμα τον αύξοντα αριθμό της κράτησης που είχε γίνει). Μια ακύρωση μπορεί να αποτύχει, αν ο αύξων αριθμός δεν αντιστοιχεί σε πραγματική κράτηση. Μια πιθανή μορφή της σχετικής συνάρτησης (που καλείται από τον client) θα μπορούσε να είναι η εξής: done = cancel_reservation(reservation_no).
3. Έκδοση εισιτηρίου για το οποίο έχει προηγηθεί κράτηση (ο αύξων αριθμός της οποίας θα δίνεται σαν όρισμα). Μια επιτυχημένη έκδοση εισιτηρίου θα πρέπει να δίνει σαν αποτέλεσμα έναν αύξοντα αριθμό. Αντίθετα σε περίπτωση αποτυχίας θα πρέπει να δίνεται το κατάλληλο μήνυμα λάθους. Μια πιθανή μορφή της σχετικής συνάρτησης (που καλείται από την πλευρά του client) θα μπορούσε να είναι η εξής: ticket_no = issue_ticket(reservation_no).
4. Παρουσίαση κρατήσεων για κάποια πτήση. Η παρουσίαση μπορεί να είναι *συνοπτική* (οπότε θα παρουσιάζεται μόνο ο συνολικός αριθμός ελεύθερων θέσεων, ο αριθμός κρατήσεων και ο αριθμός των εκδοθέντων εισιτηρίων για κάποια πτήση), ή *λεπτομερής* (οπότε θα παρουσιάζονται όλες οι κρατήσεις και εκδόσεις εισιτηρίων για κάποια πτήση). Μια πιθανή μορφή της σχετικής συνάρτησης (που καλείται από τον client) θα μπορούσε να είναι η εξής: message = show_info(FULL/BRIEF, flight_no), όπου message είναι ένα μεγάλο string που περιέχει την «κωδικοποιημένη» / «συμπυκνωμένη» απάντηση του server.

Οι λειτουργίες αυτές θα εξυπηρετούνται από τον server ο οποίος χρειάζεται να διατηρεί κάποιες δομές δεδομένων (οι οποίες θα πρέπει να ενημερώνονται σε κάθε δοσοληψία) με πληροφορίες σχετικές με τις πτήσεις και τις κρατήσεις. Αυτές είναι:

Ο αριθμός των τρεχόντων κρατήσεων και ο αριθμός των εκδοθέντων εισιτηρίων.

Για κάθε επιτυχή δοσοληψία με κάποιο client, τον αύξοντα αριθμό της δοσοληψίας (για τον οποίο θα πρέπει ο server να εξασφαλίζει πως είναι μοναδικός), τον τύπο της (αν δηλαδή είναι κράτηση θέσης ή έκδοση εισιτηρίου), τον αριθμό της πτήσης και το αντίστοιχο πλήθος θέσεων.

Η παραπάνω εφαρμογή θα πρέπει να υλοποιηθεί σε 2 διαφορετικά περιβάλλοντα:

1. Light Weight Process Library, και
2. MPI ή PVM

Εργασία 4

a. Κατανεμημένος Αλγόριθμος Αμοιβαίου Αποκλεισμού

Να υλοποιηθεί ο κατανεμημένος αλγόριθμος επίτευξης αμοιβαίου αποκλεισμού των Ricart & Agrawala. Ζητείται να υλοποιήσετε τον αλγόριθμο αυτό σε MPI. Στο σύστημα θα πρέπει να υπάρχουν N διεργασίες, οι οποίες μπορούν να τρέχουν σε διαφορετικούς κόμβους. Κάθε διεργασία πρέπει να εκδηλώνει την επιθυμία να εισέλθει στην κρίσιμη περιοχή της TIMES φορές, και αφού οι αιτήσεις της πραγματοποιηθούν πρέπει να τερματίζει. Οι διεργασίες θα πρέπει να εκτελούν ένα μη κρίσιμο τμήμα (το οποίο θα προσομοιώνεται με χρήση της sleep) πριν αποστείλουν κάθε αίτηση εισόδου στο κρίσιμο τμήμα τους.

Προσέξτε ότι ο αλγόριθμος αυτός χρησιμοποιεί τον αλγόριθμο του Lamport για την παροχή χρονικών ενδείξεων, τον οποίο επίσης θα πρέπει να υλοποιήσετε.

Το πρόγραμμα σας θα πρέπει να τυπώνει ή να καταγράφει σε ένα αρχείο κατάλληλη πληροφορία ώστε να μπορεί να ελεγχθεί η ορθότητά του.

b. Εντοπισμός Αδιεξόδου σε Κατανεμημένο Υπολογισμό

Σας ζητείται να υλοποιήσετε έναν μηχανισμό ανίχνευσης αδιεξόδου που βασίζεται στην παρακολούθηση του συστήματος από μια διεργασία παρατήρησης (monitor).

Εκτός από τη διεργασία παρατήρησης, το πρόγραμμα σας θα πρέπει να δημιουργεί ένα σύνολο τυπικών διεργασιών που αιτούνται των πόρων του συστήματος. Κάθε μια από τις διεργασίες αυτές καταναλώνει ένα αρχείο που καθορίζει τις διεργασίες 1,..., N στις οποίες η συγκεκριμένη διεργασία στέλνει αιτήσεις. Μία αίτηση έχει τη μορφή req i, όπου i είναι ο αριθμός της διεργασίας στην οποία στέλνεται η αίτηση. Όταν μια τυπική διεργασία παίρνει ένα μήνυμα αίτησης θα πρέπει να ενημερώνει τη διεργασία παρακολούθησης.

Η διεργασία παρακολούθησης διατηρεί γράφο και ελέγχει αν σε αυτόν υπάρχει κύκλος. Στην περίπτωση που το σύστημα βρίσκεται πράγματι σε αδιέξοδο, η διεργασία παρακολούθησης (εκτός από το να μας ενημερώνει με μηνύματα στην οθόνη) αποφασίζει να το επιλύσει στέλνοντας ειδικό μήνυμα σε μία από τις διεργασίες που έχουν εμπλακεί στο αδιέξοδο.

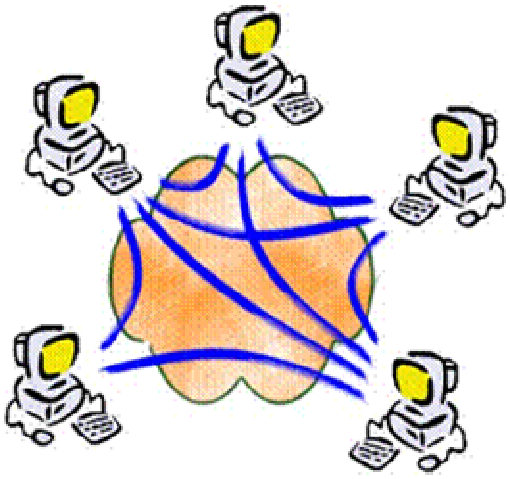
Η διεργασία αυτή αφού απεμπλακεί με τη σειρά της, πρέπει να προχωρά στην επόμενη αίτηση τυπώνοντας ένα μήνυμα λάθους.

Θα πρέπει να χρησιμοποιήσετε τη βιβλιοθήκη MPI (ή JAVA threads) για τη δημιουργία και την επικοινωνία των διεργασιών.

Εργασία 5

Ένα διομότιμο (Peer-to-Peer ή P2P) σύστημα είναι ένα κατανεμημένο σύστημα (Σχήμα 1) που αποτελείται από κόμβους που είναι ομότιμοι (δηλαδή κόμβους που έχουν την ίδια λειτουργικότητα).

Θεωρούμε ένα διομότιμο σύστημα στο οποίο το πλήθος των κόμβων που συμμετέχουν κάθε χρονική στιγμή είναι αυθαίρετο αλλά όχι μεγαλύτερο από 2^m, όπου m είναι κάποιος ακέραιος. Θεωρούμε πως κάθε κόμβος που συμμετέχει στο σύστημα έχει ένα **μοναδικό** αναγνωριστικό, δηλαδή χαρακτηρίζεται από κάποιον ακέραιο στο διάστημα από 0 έως 2^m-1. Έτσι, όλοι οι κόμβοι μπορούν να διαταχθούν, σύμφωνα με το αναγνωριστικό τους, πάνω σε έναν λογικό δακτύλιο που αναπαριστά το σύνολο τιμών {0, …, 2^m-1}. Ένα διομότιμο σύστημα εξελίσσεται δυναμικά, δηλαδή υποστηρίζει την εισαγωγή και την αποχώρηση κόμβων.

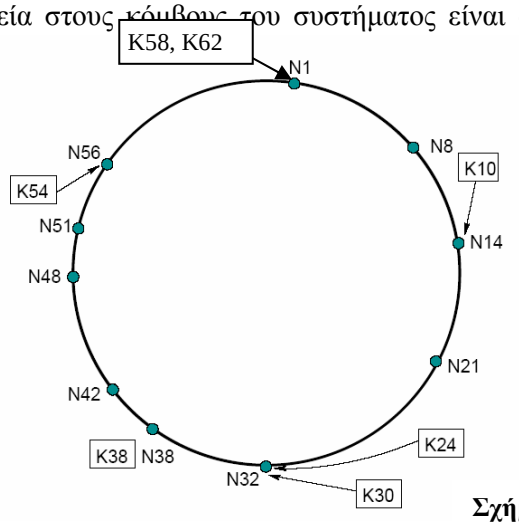


Σχήμα 1: Ένα Διομότιμο Σύστημα

Στο Σχήμα 2 παρουσιάζεται ένα διομότιμο σύστημα για το οποίο ισχύει ότι m=6 (δηλαδή ο μέγιστος αριθμός κόμβων που μπορούν να συμμετέχουν στο σύστημα είναι 2^6 = 64). Την τρέχουσα χρονική στιγμή μόνο 10 κόμβοι συμμετέχουν στο σύστημα, οι οποίοι έχουν αναγνωριστικά 1, 8, 14, 21, 32, 38, 42, 48, 51 και 56. Οι κόμβοι αυτοί, στο σχήμα εμφανίζονται ως N1, N8, N14, N21, N32, N38, N42, N48, N51 και N56, αντίστοιχα.

Μια από τις σημαντικότερες εφαρμογές ενός συστήματος P2P είναι η κατανεμημένη αποθήκευση και διαχείριση αρχείων. Θεωρούμε πως κάθε αρχείο έχει (όπως και κάθε κόμβος) ένα μοναδικό αναγνωριστικό στο σύνολο {0,…,2^m-1}. Έστω N_{min} και N_{max} οι κόμβοι με το μικρότερο και το μεγαλύτερο, αντίστοιχα, αναγνωριστικό στο δακτύλιο. Ο κανόνας σύμφωνα με τον οποίο αποθηκεύονται τα αρχεία στους κόμβους του συστήματος είναι ο ακόλουθος:

«(α) Κάθε κόμβος ≠ N_{min} αποθηκεύει τα αρχεία των οποίων τα αναγνωριστικά είναι μεταξύ του αναγνωριστικού του κόμβου και του αναγνωριστικού του προηγούμενου κόμβου στο δακτύλιο και (β) ο Nmin αποθηκεύει κάθε αρχείο με αναγνωριστικό K τέτοιο ώστε $N_{max} \leq K \leq 2^m$-1 ή $0 \leq K \leq N_{min}$». *(1)*



Σχήμα 2

Για παράδειγμα, στο Σχήμα 2, έχουν αποθηκευτεί 7 αρχεία στο σύστημα, τα οποία έχουν αναγνωριστικά 10, 24, 30, 38, 54, 58 και 62. Το αρχείο με αναγνωριστικό 10 αποθηκεύεται στον κόμβο με αναγνωριστικό 14 (αφού το 10 είναι μεταξύ του αναγνωριστικού 8 που έχει ο προηγούμενος κόμβος του 14 στο σύστημα και του 14), τα αρχεία με αναγνωριστικά 24 και 30 αποθηκεύονται στον κόμβο με αναγνωριστικό 32, το αρχείο με αναγνωριστικό 38 αποθηκεύεται στον κόμβο με αναγνωριστικό 38, εκείνο με αναγνωριστικό 54 αποθηκεύεται στον κόμβο με αναγνωριστικό 56, ενώ τα αρχεία με αναγνωριστικά 58 και 62 αποθηκεύονται στον κόμβο με αναγνωριστικό 1.

Αν στο σύστημα εισαχθεί κόμβος με αναγνωριστικό 26, ο κόμβος 32 θα συνεχίσει να αποθηκεύει μόνο το αρχείο με αναγνωριστικό 30 ενώ εκείνο με αναγνωριστικό 24 θα αποθηκευθεί στο νέο κόμβο (δηλαδή σε εκείνο με αναγνωριστικό 26) ώστε να ισχύει η ιδιότητα (1).

Ζητείται να υλοποιηθεί το διομότιμο σύστημα που περιγράφτηκε παραπάνω. Πιο συγκεκριμένα, το πρόγραμμα σας θα πρέπει να χρησιμοποιεί το MPI για να υλοποιεί ένα διομότιμο σύστημα που διέπεται από τους κανόνες που περιγράφηκαν παραπάνω.

Τα γεγονότα που μπορούν να συμβούν στο σύστημα είναι της μορφής:

- J <id>: Γεγονός τύπου Join το οποίο σηματοδοτεί την εισαγωγή ενός κόμβου στο σύστημα με αναγνωριστικό <id>. Ο κόμβος πρέπει να τοποθετηθεί στη σωστή θέση της λίστας των κόμβων. Επίσης, η λίστα των αρχείων του πρέπει να αρχικοποιηθεί κατάλληλα. Πιο συγκεκριμένα, τα αρχεία που βρίσκονται στη λίστα αρχείων του επομένου κόμβου στο δακτύλιο μοιράζονται ώστε να εξακολουθήσει να ισχύει η συνθήκη (1) (δηλαδή κάθε κόμβος να έχει στη λίστα των αρχείων του εκείνα τα αρχεία για τα οποία ισχύει πως το αναγνωριστικό τους είναι μεταξύ του αναγνωριστικού του κόμβου και του αναγνωριστικού του προηγουμένου του κόμβου στο δακτύλιο). Για παράδειγμα, έστω ο κόμβος με αναγνωριστικό 50 και έστω ότι ο προηγούμενός του στη λίστα των κόμβων είναι ο 20. Έστω ότι ο 50 έχει στη λίστα των αρχείων του τα αρχεία με αναγνωριστικά 25, 33, 36, 41, 46, 49. Αν στο σύστημα εισαχθεί ο κόμβος με αναγνωριστικό 40, ο 40 είναι πλέον ο προηγούμενος του 50 και τα αρχεία με αναγνωριστικά 25, 33 και 36 πρέπει να μεταφερθούν από τη λίστα αρχείων του 50 στη λίστα αρχείων του 40.
- I <id> <tp>: Γεγονός τύπου Insert το οποίο σηματοδοτεί την εισαγωγή ενός αρχείου με αναγνωριστικό <id> τύπου <tp> στο σύστημα.
- L <id>: Γεγονός τύπου Leave το οποίο σηματοδοτεί την αποχώρηση του κόμβου με αναγνωριστικό <id> από το σύστημα. Ο κόμβος θα πρέπει να παραχωρήσει όλα τα αρχεία του στον επόμενό του κόμβο στη λίστα των κόμβων.
- F <id>: Γεγονός τύπου Find το οποίο σηματοδοτεί την αναζήτηση ενός αρχείου με αναγνωριστικό <id> στο σύστημα.
- D <id>: Γεγονός τύπου Delete το οποίο σηματοδοτεί τη διαγραφή του αρχείου με αναγνωριστικό <id> από το σύστημα.

Για κάθε κόμβο θα πρέπει να υπάρχει ένα αρχείο που να περιγράφει τα γεγονότα που θα πρέπει να εκτελέσει ο κόμβος. Κάθε κόμβος θα πρέπει να αποθηκεύει κατάλληλη πληροφορία για τα γεγονότα που εκτελεί ή του ζητούνται να εκτελέσει ώστε να μπορεί να γίνει αποσφαλμάτωση του προγράμματος και να ελεγχθεί η ορθότητά του.