Ιονιο Πανεπιστημιο

Ο αλγόριθμος Quicksort: Υλοποίηση Thread Pool με POSIX Threads.

Κωνσταντίνος Ξυπολιτόπουλος

11 Μαΐου 2018

1 Εισαγωγη

Ζητούμενο: Στην παρούσα άσκηση ζητείται να υλοποιήσετε τον αλγόριθμο Quicksort με μία δεξαμενή νημάτων (thread pool). Στη δεξαμενή αυτή θα υπάρχει ένας σταθερός μικρός αριθμός νημάτων (π.χ. 4), τα οποίαθα δημιουργούνται στην αρχή του προγράμματος και θα τερματίζουν όταν θα έχει ολοκληρωθεί η ταξινόμηση. Τα νήματα θα αναλαμβάνουν πακέτα εργασίας από μια παγκόσμια (global) ουρά εργασιών.

Η υλοποίηση του παραπάνω ζητουμένου έχει 3 κεντρικά προβλήματα:

- Την δημιουργία και σωστή διαχείρηση των διαφορετικών νημάτων που θα εκτελούν παράλληλα τον αλγόριθμο Quicksort.
- Την υλοποίηση της ουράς που θα περιέχει τις εργασίες (tasks) που πρέπει να εκτελέσουν τα διάφορα νήματα ανεξάρτητα.
- Τον κεντρικό κώδικα ελέγχου ώστε να καταλάβουμε πότε έχει τελειώσει ο αλγόριθμος και να μπορέσουμε έπειτα να ¨κλείσουμε' όλα τα νήματα ομαλά.

2 Conditional Variables & Mutexes

Έχουμε δύο μεταβλητές συνθήκης conditional variables, η πρώτη, (job in), έχει την δουλειά του να ενημερώνει τα νήματα κάθε φορά που μπαίνουν καινούργια tasks στην ουρά, κάθε νήμα εκτέλεσης που περιμένει (pthread cond wait) σε αυτή περιμένει για μια καινούργια δουλεία. Η δεύτερη, (job out), έχει να κάνει με τον τερματισμό του προγράμματος, κάθε φορά που ένα νήμα τελειώνει ένα task από την ουρά στέλνει σήμα μέσω της μεταβλητής στο κύριο νήμα που εκτελεί την main να ελέγξει για το αν έχει τελειώσει ο αλγόριθμος.

Επίσης έχουμε μια μεταβλητή αμοιβαίου αποκλεισμού (mutual exclusion - mutex) που συνεργάζεται με τις λειτουργίες των παραπάνω δύο μεταβλητών συνθήκης για να μας δώσει πολύ μεγάλο έλεγχο γύρω από την εκτέλεση του προγράμματος μας. Κάθε φορά που κλειδώνουμε τον mutex μετά μπορούμε να χρησιμοποιήσουμε την συνάρτηση αναμονής των condition variables για να ξεκλειδώσουμε την εκτέλεση του προγράμματος, αλλά με το που η συνθήκη που περιμένουμε πάρει το σήμα της με ένα κάλεσμα στην pthread cond signal από

κάποιο άλλο νήμα, αυτόματα κλειδώνουμε πάλι την εκτέλεση του προγράμματος στο δικό μας νήμα με τον mutex.

3 Ροη Προγραμματοσ

3.1 Αρχικοποιησεισ

Αρχικά το πρόγραμμα ξεκινάει με ένα νήμα, το κύριο νήμα εκτέλεσης που εκτελέι την συνάρτηση main, δηλώνουμε ένα μονοδιάστατο πίνακα μεγέθους N στην μνήμη heap χρησιμοποιώντας την συνάρτηση malloc(αν τον δηλώναμε στην τοπική μνήμη stack θα είχαμε πρόβλημα γιατί το κάθε νήμα έχει την δική του και δεν μοιράζεται μεταξύ τους) και τον γεμίζουμε τυχαίες τιμές με την χρήση μιας ψευδοτυχαίας συνάρτησης βιβλιοθήκης. Στην συνέχεια δημιουργούμε 4 (ή μεχρι 8) νήματα και θέτουμε την συνάρτηση που θα τρέξουν, τα νήματα αρχικά περιμένουν όλα το σήμα του job in.

3.2 Νηματα

Στέλνουμε το πρώτο task (ολόκληρο τον πίνακα N θέσεων) στην ουρά εργασίας και σηματοδοτούμε ένα νήμα να ξεκινήσει, με το που πάρει κάποιο νήμα το σήμα μέσω μιας conditional variable ξεκινάει την πρώτη τμηματοποίηση (partitioning) του πίνακα σε δύο μέρη, σε σχέση με το pivot που διαλέγουμε χωρίζει τον πίνακα που του δώθηκε σε ένα τμήμα που όλα τα νούμερα έιναι μικρότερο του pivot και ένα που είναι μεγαλύτερα, έτσι ξεκινούν δύο νήματα αφού έχουμε δύο tasks στην ουρά. Κάθε φορά που κάποιο νήμα αναλάβει μία εργασία από την ουρά και η εργασία είναι για μήκος πίνακα μικρότερο μιας τιμής που θέτουμε εμείς (στην περίπτωσή μας το 10) τότε δεν καλεί την συνάρτηση partition και δεν δημιουργεί δύο καινούργιες εργασίες, απλά εκτελεί την εργασία του καλώντας την συνάρτηση insertion sort και ξανά μετά κοιμάται μέχρι να λάβει σήμα στην job in.

3.3 Αποδοση

Μέσα στο κάθε νήμα έχουμε εκτέλεση δύο συναρτήσεων που εργάζονται πάνω στην ίδια δομή δεδομένων (τον πίνακα που κάνουμε ταξινόμηση), αυτό θα μας έκανε να σκεφτούμε ότι χρειαζόμαστε να αποκλείσουμε με mutex την εργασία πάνω από ενός μόνο νήματος στον πίνακα, και θα είχαμε δίκιο, αν δουλέυαμε με κάποιους άλλους αλγορίθμους. Ο τρόπος όμως που δουλεύει ο διαχωρισμός του πίνακα σε δύο υποπίνακες και μετά σε άλλους δύο κτλ, σημαίνει ότι κανένα νήμα δεν μπορεί να δουλέψει σε περιοχή που δουλέυει το άλλο νήμα την ίδια στιγμή, οπότε η μεγάλη παρατήρηση εδώ είναι ότι δεν χρειάζεται να αποκλείσουμε την παράλληλη εργασία νημάτων πάνω στον ίδιο πίνακα όταν εκτελούν την partition και την insertionsort, αυτή η παρατήρηση είναι όλος ο λόγος για την παραλληλοποίηση της quicksort.

3.4 Τερματισμος Νηματων και Ελεγχος Λυσης

Τελικά, όταν τελειώσουν όλες οι εργασίες στην ουρά όλα τα νήματα θα μείνουν κολλημένα περιμένωντας την επόμενη job in που όμως δεν θα έρθει ποτέ, τι κάνουμε για να τερματίσουμε τα νήματα τοτε; Η απάντηση είναι ότι τρέχουμε μια συνθήκη ελέγχου τερματισμού στο κύριο νήμα, κάθε φορά που ένα νήμα τελειώνει μια εργασία στέλνει σήμα στην άλλη conditional variable την job out, αφού το κύριο νήμα είναι το μόνο που περιμένει σε αυτή την μεταβλητή, θα ξυπνάει πάντα και θα εκτελέι ένα έλεγχο για το πόσα στοιχεία έχουν ταξινομηθεί. Αν δει ότι έχουν ταξινομηθεί όλα τα στοιχεία στον πίνακα, στέλνει ένα μήνυμα και ξυπνάει αυτή την φορά όλα τα νήματα, τα νήματα αφού ξυπνήσουν, βλέπουν ότι έχουν ταξινομηθεί όλα τα στοιχεία και τερματίζουν την συνάρτηση τους, έπειτα το κύριο νήμα ενώνει (joins) όλα τα νήματα στο κεντρικό και ελέγχει την εγκυρότητα της ταξινόμησης κατα μήκος όλου του πίνακα.
Τέλος απλευθερώνουμε την μνήμη που θέσαμε για τις διάφορες δομές όπως η ουρά ελέγχου, ο δυναμικός πίνακας που ταξινομήσαμε, τα mutexes και conditional variables, και τερματίζουμε την εφαρμογή.

4 Κωδικας

Μερικά κομμάτια του κώδικα της εργασίας, όπως οι συναρτήσεις partition και insertion sort, πάρθηκαν από τα εργαστήρια του μαθήματος. Ο υπόλοιπος κώδικας γράφτηκε όλοκληρωτικά από εμένα για την παρούσα εργασία. Ο κώδικας

περιέχει βασικό σχολιασμό κάνοντας το διάβασμα της αναφοράς προαπαιτούμενο για την κατανόηση του, δεν έκρινα αναγκαίο να μπώ σε μεγαλύτερες τεχνικές λεπτομέρειες όσον αφορά τον κώδικα, όπως κάποιες συναρτήσεις που έγραψα για διευκόλυνση της εφαρμογής και καθαρότητας του κώδικα των νημάτων.