ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Λειτουργικά Συστήματα Υπολογιστών

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2024-2025

2η Εργαστηριακή Άσκηση
Συγχρονισμός

Εργαστήριο Υπολογιστικών Συστημάτων Ε.Μ.Π.

Απρίλιος 2025

Στην παρούσα εργαστηριακή άσκηση καλείστε να χρησιμοποιήσετε διαδεδομένους μηχανισμούς συγχρονισμού για να επιλύσετε προβλήματα συγχρονισμού σε πολυνηματικές εφαρμογές βασισμένες στο πρότυπο POSIX threads.

Οι μηχανισμοί που θα χρησιμοποιήσετε είναι (α) τα κλειδώματα (*mutexes*), οι σημαφόροι (*semaphores*) και οι μεταβλητές συνθήκης (*condition variables*) που ορίζονται από το POSIX, (β) οι ατομικές λειτουργίες (atomic operations), όπως ορίζονται από το υλικό και εξάγονται στον προγραμματιστή μέσω ειδικών εντολών (*builtins*) του μεταγλωττιστή GCC.

Για περισσότερες πληροφορίες για τα POSIX threads (εκτός από την πληθώρα των αποτελεσμάτων που θα σας δώσει μία αναζήτηση στο διαδίκτυο):

- David R. Butenhof, ``Programming with POSIX Threads'', Amazon link
- Το μέρος ``System Interfaces'' του προτύπου POSIX.1-2008, διαθέσιμου και σε μορφή man pages. Σε συστήματα βασισμένα στο Debian, μπορείτε να εγκαταστήσετε τα πακέτα `manpages-posix`, `manpages-posix-dev`.

Για περισσότερες πληροφορίες σχετικά με τα atomic operations, την υλοποίησή τους από τον μεταγλωττιστή GCC και τη χρήση τους σε ένα μεγάλο project όπως ο πυρήνας του Linux, δείτε:

- Το μέρος Built-in functions for atomic memory access από το εγχειρίδιο του GCC.
- Το μέρος atomic ops της τεκμηρίωσης του πυρήνα του Linux.

1 Συγχρονισμός σε υπάρχοντα κώδικα

Εξοικειωθείτε με τη χρήση των POSIX threads μελετώντας το υπόδειγμα `pthread-test.c` που σας δίνεται.

Σας δίνεται το πρόγραμμα `simplesync.c`, το οποίο λειτουργεί ως εξής: Αφού αρχικοποιήσει μια μεταβλητή `val = 0`, δημιουργεί δύο νήματα τα οποία εκτελούνται ταυτόχρονα: το πρώτο νήμα αυξάνει N φορές την τιμή της μεταβλητής `val` κατά 1, το δεύτερο τη μειώνει N φορές κατά 1. Τα νήματα δεν συγχρονίζουν την εκτέλεσή τους.

Ζητούνται τα εξής:

- Χρησιμοποιήστε το παρεχόμενο `Makefile` για να μεταγλωττίσετε και να τρέξετε το πρόγραμμα. Τι παρατηρείτε; Γιατί;
- Μελετήστε πώς παράγονται δύο διαφορετικά εκτελέσιμα `simplesync-atomic`, `simplesync-mutex`, από το ίδιο αρχείο πηγαίου κώδικα `simplesync.c`.

- Επεκτείνετε τον κώδικα του `simplesync.c` ώστε η εκτέλεση των δύο νημάτων στο εκτελέσιμο `simplesync-mutex` να συγχρονίζεται με χρήση POSIX mutexes. Επιβεβαιώστε την ορθή λειτουργία του προγράμματος.
- Επεκτείνετε τον κώδικα του `simplesync.c` ώστε η εκτέλεση των δύο νημάτων στο εκτελέσιμο `simplesync-atomic` να συγχρονίζεται με χρήση ατομικών λειτουργιών του GCC. Επιβεβαιώστε την ορθή λειτουργία του προγράμματος.

Σημείωση: Όλα τα αρχεία κώδικα που δίνονται για την άσκηση βρίσκονται στον κατάλογο `/home/oslab/code/sync`.
Μελετήστε προσεκτικά τον κώδικα που σας δίνεται πριν ξεκινήσετε!

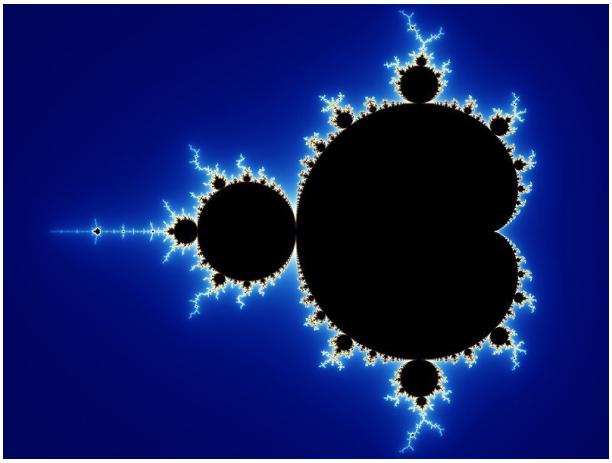
Ερωτήσεις:

1. Χρησιμοποιήστε την εντολή `time(1)` για να μετρήσετε το χρόνο εκτέλεσης των εκτελέσιμων. Πώς συγκρίνεται ο χρόνος εκτέλεσης των εκτελέσιμων που εκτελούν συγχρονισμό, σε σχέση με το χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό; Γιατί;
2. Ποια μέθοδος συγχρονισμού είναι γρηγορότερη, η χρήση ατομικών λειτουργιών ή η χρήση POSIX mutexes; Γιατί;
3. Σε ποιες εντολές του επεξεργαστή μεταφράζεται η χρήση ατομικών λειτουργιών του GCC στην αρχιτεκτονική για την οποία μεταγλωττίζετε; Χρησιμοποιήστε την παράμετρο `-S` του GCC για να παράγετε τον ενδιάμεσο κώδικα Assembly, μαζί με την παράμετρο `-g` για να συμπεριλάβετε πληροφορίες γραμμών πηγαίου κώδικα (π.χ., ``.loc 1 63 0''), οι οποίες μπορεί να σας διευκολύνουν. Δείτε την έξοδο της εντολής `make` για τον τρόπο μεταγλώττισης του `simplesync.c`.
4. Σε ποιες εντολές μεταφράζεται η χρήση POSIX mutexes στην αρχιτεκτονική για την οποία μεταγλωττίζετε; Παραθέστε παράδειγμα μεταγλώττισης λειτουργίας `pthread_mutex_lock()` σε Assembly, όπως στο προηγούμενο ερώτημα.

2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

Σας δίνεται πρόγραμμα που υπολογίζει και εξάγει στο τερματικό εικόνες του συνόλου του Mandelbrot (Σχ. 1).

Το σύνολο του Mandelbrot ορίζεται ως το σύνολο των σημείων c του μιγαδικού επιπέδου, για τα οποία η ακολουθία $z_{n+1} = {z_n}^2 + c$ είναι φραγμένη. Ένας



Σχήμα 1: Πηγή εικόνας

απλός αλγόριθμος για τη σχεδίασή του είναι ο εξής: Ξεκινάμε αντιστοιχίζοντας την επιφάνεια σχεδίασης σε μια περιοχή του μιγαδικού επιπέδου. Για κάθε pixel παίρνουμε τον αντίστοιχο μιγαδικό c και υπολογίζουμε επαναληπτικά την ακολουθία $z_{n+1} = {z_n}^2 + c, z_0 = 0$ έως ότου $|z_n| > 2$ ή το n ξεπεράσει μια προκαθορισμένη τιμή. Ο αριθμός των επαναλήψεων που απαιτήθηκαν αντιστοιχίζεται ως χρώμα του συγκεκριμένου pixel.

Στο `mandel.c` σας δίνεται ένα πρόγραμμα που υπολογίζει και σχεδιάζει το σύνολο Mandelbrot σε τερματικό κειμένου, χρησιμοποιώντας χρωματιστούς χαρακτήρες. Για τη λειτουργία του βασίζεται στη βιβλιοθήκη `mandel-lib.{c, h}`. Η βιβλιοθήκη υλοποιεί τις εξής συναρτήσεις:

- `mandel_iterations_at_point()`: Υπολογίζει το χρώμα ενός σημείου `(x, y)` με βάση τον παραπάνω αλγόριθμο.
- `set_xterm_color()`: θέτει το χρώμα των χαρακτήρων που εξάγονται στο τερματικό.

Η έξοδος του προγράμματος είναι ένα μπλοκ χαρακτήρων, διαστάσεων `x_chars` στηλών και `y_chars` γραμμών. Το πρόγραμμα καλεί επαναληπτικά την `compute_and_output_mandel_line()` για την εκτύπωση του μπλοκ γραμμή προς γραμμή.

Ζητείται η επέκταση του προγράμματος `mandel.c` έτσι ώστε ο υπολογισμός να κατανέμεται σε `NTHREADS` νήματα POSIX, ενδεικτική τιμή 3. Η κατανομή του υπολογιστικού φόρτου γίνεται ανά σειρά: Για n νήματα, το i-ιοστό (με $i = 0, 1, 2, \ldots, n-1$) αναλαμβάνει τις σειρές $i, i+n, i+2\times n, i+3\times n, \ldots$

Ζητούνται 2 εκδοχές του προγράμματος όπου ο απαραίτητος *συγχρονισμός* των νημάτων θα γίνεται:

1. **με σημαφόρους** που παρέχονται από το πρότυπο POSIX, συμπεριλαμβάνοντας το αρχείο επικεφαλίδας `<semaphore.h>`. Δείτε περισσότερα στα manual pages `sem_overview(7)`,`sem_wait(3)`, `sem_post(3)`.
2. **με μεταβλητές συνθήκης (*condition variables*)** που παρέχονται από το πρότυπο POSIX, συμπεριλαμβάνοντας το αρχείο επικεφαλίδας `<pthread.h>`. Δείτε περισσότερα στα manual pages `pthread_cond_wait(3)`, `pthread_cond_broadcast(3)`, `pthread_cond_signal(3)`

Σε κάθε περίπτωση η τιμή `NTHREADS` θα δίνεται κατά το χρόνο εκτέλεσης του προγράμματος, ως όρισμα στη γραμμή εντολών.

> **Σημείωση**: Αν το τερματικό σας αφεθεί με λάθος χρώμα στο κείμενο λόγω της εκτέλεσης του προγράμματος [π.χ. μωβ χαρακτήρες σε μαύρο φόντο], χρησιμοποιήστε την εντολή `reset` για να το επαναφέρετε στην αρχική ρύθμισή του.

Ερωτήσεις:

1. Πόσοι σημαφόροι χρειάζονται για το σχήμα συγχρονισμού που υλοποιείτε;
2. Πόσος χρόνος απαιτείται για την ολοκλήρωση του σειριακού και του παράλληλου προγράμματος με δύο νήματα υπολογισμού; Χρησιμοποιήστε την εντολή `time(1)` για να χρονομετρήσετε την εκτέλεση ενός προγράμματος, π.χ., `time sleep 2`. Για να έχει νόημα η μέτρηση, δοκιμάστε σε ένα μηχάνημα που διαθέτει επεξεργαστή δύο πυρήνων. Χρησιμοποιήστε την εντολή `cat /proc/cpuinfo` για να δείτε πόσους υπολογιστικούς πυρήνες διαθέτει κάποιο μηχάνημα.
3. Πόσες μεταβλητές συνθήκης χρησιμοποιήσατε στη δεύτερη εκδοχή του προγράμματος σας? Αν χρησιμοποιηθεί μια μεταβλητή πως λειτουργεί ο συγχρονισμός και ποιο πρόβλημα επίδοσης υπάρχει?
4. Το παράλληλο πρόγραμμα που φτιάξατε, εμφανίζει επιτάχυνση; Αν όχι, γιατί; Τι πρόβλημα υπάρχει στο σχήμα συγχρονισμού που έχετε υλοποιήσει; *Υπόδειξη:* Πόσο μεγάλο είναι το κρίσιμο τμήμα; Χρειάζεται να περιέχει και τη φάση υπολογισμού και τη φάση εξόδου κάθε γραμμής που παράγεται;
5. Τι συμβαίνει στο τερματικό αν πατήσετε Ctrl-C ενώ το πρόγραμμα εκτελείται; σε τι κατάσταση αφήνεται, όσον αφορά το χρώμα των γραμμάτων; Πώς θα μπορούσατε να επεκτείνετε το `mandel.c` σας ώστε να εξασφαλίσετε ότι ακόμη κι αν ο χρήστης πατήσει Ctrl-C, το τερματικό θα επαναφέρεται στην προηγούμενη κατάστασή του;