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

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 3: Συγχρονισμός

1 Ασκήσεις

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.

1.1 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.

1.2 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 απλός αλγόριθμος για

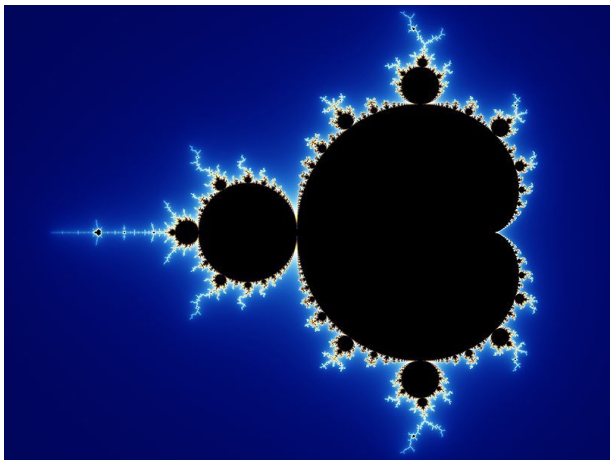

Figure 1: Εικόνα από το http://en.wikipedia.org/wiki/Mandelbrot_set

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 $i, i+n, i+2 \times n, i+3 \times n, \ldots$

Ο απαραίτητος συγχρονισμός των νημάτων θα γίνει 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)`.

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. 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;
4. 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;

2 Προαιρετική άσκηση (επιπλέον βαθμοί: 0)

2.1 Επίλυση προβλήματος συγχρονισμού

Καλείστε να δώσετε σχήμα συγχρονισμού για την επίλυση του εξής προβλήματος:

Σε ένα νηπιαγωγείο (“Kindergarten”), βρίσκονται παιδιά και δάσκαλοι. Οι κανονισμοί επιβάλλουν να είναι πάντα παρών ένας δάσκαλος ανά `R` παιδιά, όπου `R` ακέραιος αριθμός (π.χ. αναλογία παιδιών/δασκάλων 3:1), ώστε να εξασφαλίζεται η σωστή επίβλεψη των παιδιών.

Τα παιδιά κι οι δάσκαλοι υλοποιούνται με χρήση δύο διαφορετικών τύπων νημάτων, T_C, T_T αντίστοιχα. Κάθε τύπος νήματος εκτελεί τον εξής κώδικα:

```
void Teacher()
{
    for (;;) {
        teacher_enter();
        ...critical section...
        teacher_exit();
        go_home_and_rest();
    }
}
```

```
void Child()
{
    for (;;) {
        child_enter();
        ...critical section...
        child_exit();
        go_home_and_rest();
    }
}
```

Υπάρχουν `N` νήματα συνολικά, `C` από τα οποία προσομοιώνουν παιδιά, εκτελώντας τη συνάρτηση `Child()`, ενώ τα υπόλοιπα `C-N` προσομοιώνουν δασκάλους, εκτελώντας τη συνάρτηση `Teacher()`.

Σας ζητείται να υλοποιήσετε κατάλληλο σχήμα συγχρονισμού ορίζοντας τις συναρτήσεις `teacher_enter()`, `teacher_exit()`, `child_enter()`, `child_exit()`. Μπορείτε να χρησιμοποιήσετε κλειδώματα και μεταβλητές συνθήκης (mutexes/locks, condition variables).
hint: Αρκεί η χρήση μίας μεταβλητής συνθήκης.

Για να εξοικειωθείτε με το πρόβλημα σας δίνεται υλοποίησή του με POSIX threads στο αρχείο `kgarten.c`. Το νηπιαγωγείο περιγράφεται από στιγμιότυπο της δομής `kgarten_struct`, το οποίο είναι κοινό για όλα τα νήματα.

Οι τιμές `N`, `C`, `R` δίνονται ως ορίσματα στη γραμμή εντολών.

Βήματα:

1. Μελετήστε τον κώδικα του `kgarten.c`.
2. Εκτελέστε το και παρατηρήστε πώς η έλλειψη συγχρονισμού ανάμεσα στα νήματα οδηγεί σε καταστροφικά αποτελέσματα. Δοκιμάστε με διαφορετικές παραμέτρους για τις τιμές εισόδου, π.χ. `N = 10`, `C = 7`, `R = 2` ή `R = 3`.
3. Υλοποιήστε κατάλληλο σχήμα συγχρονισμού έτσι ώστε το πρόγραμμα να εκτελείται χωρίς προβλήματα. Επεκτείνετε με επιπλέον πεδία τη δομή `kgarten_struct` και συμπληρώστε το σώμα των συναρτήσεων `teacher_enter()`, `teacher_exit()`, `child_enter()`, `child_exit()`, χωρίς να αλλάξετε το υπόλοιπο πρόγραμμα.
4. Δοκιμάστε το σχήμα συγχρονισμού σας και πειραματιστείτε με διαφορετικές τιμές των `N`, `C`, `R`. Αν θέλετε μπορείτε να πειράξετε και τους ενδιάμεσους χρόνους αναμονής (τιμές στις κλήσεις `usleep()`).
5. Μελετήστε θεωρητικά το σχήμα συγχρονισμού που προτείνετε και εξετάστε αν λειτουργεί σωστά (π.χ. δεν οδηγεί σε αδιέξοδο) για οποιαδήποτε αλληλουχία εκτέλεσης των νημάτων.

> Σημείωση: Το `kgarten.c` προσπαθεί να εντοπίσει προβλήματα στο σχήμα συγχρονισμού, μετρώντας το πλήθος και το είδος των νημάτων μέσα στο κρίσιμο νήμα. Ωστόσο, ο τρόπος λειτουργίας του εισάγει κατάσταση συναγωνισμού (race) και δεν είναι αξιόπιστος. Το πρόγραμμα `kgarten.c` σας δίνεται για να εξοικειωθείτε με το πρόβλημα, να μελετήσετε τον τρόπο υλοποίησης των νημάτων, και να μπορείτε να δοκιμάζετε γρήγορα διαφορετικά σχήματα συγχρονισμού. Ωστόσο, η σωστή εκτέλεσή του με το σχήμα συγχρονισμού σας δεν αρκεί για να αποδείξει την ορθότητά του, καθώς είναι πιθανό να υπάρχουν προβληματικές καταστάσεις τις οποίες δεν μπορεί να εντοπίσει.

Ερωτήσεις:

1. Έστω ότι ένας από τους δασκάλους έχει αποφασίσει να φύγει, αλλά δεν μπορεί ακόμη να το κάνει καθώς περιμένει να μειωθεί ο αριθμός των παιδιών στο χώρο (κρίσιμο τμήμα). Τι συμβαίνει στο σχήμα συγχρονισμού σας για τα νέα παιδιά που καταφτάνουν και επιχειρούν να μπουν στο χώρο;
2. Υπάρχουν καταστάσεις συναγωνισμού (races) στον κώδικα του `kgarten.c` που επιχειρεί να επαληθεύσει την ορθότητα του σχήματος συγχρονισμού που υλοποιείτε; Αν όχι, εξηγήστε γιατι. Αν ναι, δώστε παράδειγμα μιας τέτοιας κατάστασης.

3 Εξέταση άσκησης και αναφορά

Η προθεσμία για την εξέταση της άσκησης έχει ανακοινωθεί στη σελίδα του μαθήματος.

Μετά την εξέταση η κάθε ομάδα θα πρέπει να συντάξει μια (σύντομη) αναφορά και να τη στείλει μέσω email στους βοηθούς εργαστηρίου. Η προθεσμία για την αναφορά είναι μια εβδομάδα μετά την προθεσμία εξέτασης της άσκησης.

Η αναφορά αυτή θα περιέχει:

- Τον πηγαίο κώδικα (source code) των ασκήσεων.
- Την έξοδο εκτέλεσης των προγραμμάτων για διάφορες παραμέτρους, ενδεικτικά.
- Σύντομες απαντήσεις στις ερωτήσεις.

4 Προαιρετικές ερωτήσεις (επιπλέον βαθμοί: 0)

1. Σκιαγραφήστε υλοποίηση σημαφόρων χρησιμοποιώντας Ελεγκτές/Παρακολουθητές (Monitors)

2. Εκτελέστε το πρόγραμμα που προκύπτει από το αρχείο `rand-fork.c`. Τι παρατηρείτε; Γιατί συμβαίνει;

3. Περιγράψτε σχήμα συγχρονισμού στο `kgarten.c` που επιτρέπει να μπαίνουν στο κρίσιμο τμήμα τα νέα παιδιά που φτάνουν με μεγαλύτερη πιθανότητα από έναν δάσκαλο που περιμένει να μειωθεί ο αριθμός των παιδιών στο χώρο και να φύγει. Ποιες αλλαγές χρειάζονται στον κώδικά σας;