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

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2011-2012

Άσκηση 3: Συγχρονισμός Διεργασιών

1 Ασκήσεις

1.1 Υλοποίηση σημαφόρων με σωληνώσεις του UNIX

Ζητείται η υλοποίηση μιας βιβλιοθήκης σημαφόρων (`pipesem`) βασισμένης στο μηχανισμό σωληνώσεων του UNIX. Η βιβλιοθήκη θα αποτελείται από τα αρχεία `pipesem.h` και `pipesem.c`. Σας δίνεται το `pipesem.h` που ορίζει τη διεπαφή (API) της βιβλιοθήκης. Αυτή είναι:

- Ο τύπος `struct pipesem`, που ορίζει ένα σημαφόρο ως τα δύο άκρα ενός pipe.
- Οι συναρτήσεις `pipesem_{init, destroy, wait, signal}()` για τις λειτουργίες αρχικοποίησης, καταστροφής, πράξης *wait* και πράξης *signal* σε σημαφόρο, αντίστοιχα.

Η πράξη *wait* αντιστοιχίζεται σε `read()` από τη σωλήνωση, η πράξη *signal* σε `write()`. Για τον έλεγχο της καλής λειτουργίας της βιβλιοθήκης σας, θα χρησιμοποιήσετε το παράδειγμα `pipesem-test.c` που δίνεται.

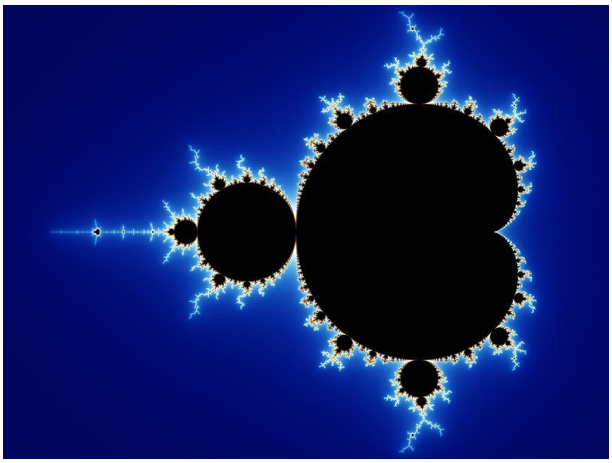
Σημείωση: Το αρχείο `pipesem-test.c` μαζί με όλα τα αρχεία κώδικα που δίνονται για την άσκηση βρίσκεται στον κατάλογο `/home/oslab/code/sync`.
Πριν ξεκινήσετε την υλοποίηση μελετήστε τον κώδικα που δίνεται!

Ερωτήσεις:

1. Τι ακριβώς κάνει το `pipesem-test.c`; Πόσες διεργασίες τρέχουν και πώς αυτές συγχρονίζονται;
2. Ποια είναι η βασική αρχή λειτουργίας της βιβλιοθήκης `pipesem.c`; Πώς εξασφαλίζεται ότι η πράξη $wait$ σε σημαφόρο μηδενικής ή αρνητικής τιμής μπλοκάρει; Τι σημαίνει αυτό για την υφιστάμενη σωλήνωση του UNIX;
3. Θα μπορούσε αυτός ο τρόπος υλοποίησης να οδηγήσει σε λιμοκτονία (starvation); Πώς επηρεάζεται η λειτουργία του από τον τρόπο με τον οποίο το ΛΣ υλοποιεί την ανάγνωση από pipes, όταν πολλές διεργασίες χρειάζεται να περιμένουν σε άδεια σωλήνωση;

1.2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

Σας δίνεται πρόγραμμα που υπολογίζει και εξάγει στο τερματικό εικόνες του συνόλου του Mandelbrot:



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

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 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 του `mandel.c` έτσι ώστε ο υπολογισμός να μοιράζεται σε `NCHILDREN`, ενδεικτική τιμή 3, διεργασίες. Η κατανομή του υπολογιστικού φόρτου γίνεται ανά σειρά: Για n διεργασίες, η i-ιοστή (με $i = 0, 1, 2, \ldots$) αναλαμβάνει τις σειρές $i, i + n, i + 2 \times n, i + 3 \times n, \ldots$

Ο απαραίτητος *συγχρονισμός* των διεργασιών θα γίνει με σημαφόρους που παρέχονται από τη βιβλιοθήκη `pipesem` του προηγούμενου ερωτήματος.

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 διεργασίες υπολογισμού; χρησιμοποιήστε την εντολή `time` 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 επεκτείνεται το `mandel.c` σας ώστε να εξασφαλίσετε ότι ακόμη κι αν ο χρήστης πατήσει Ctrl-C, το τερματικό θα επαναφέρεται στην προηγούμενη κατάστασή του;

1.3 Ταυτόχρονη πρόσβαση σε μοιραζόμενους πόρους

Δίνεται το πρόγραμμα `procs-shm.c`, το οποίο δημιουργεί τρεις διεργασίες P_A, P_B και P_C. Οι διεργασίες εκτελούν προσβάσεις σε μια *μοιραζόμενη* ακέραια μεταβλητή n, σε ατέρμονα βρόχο, ως εξής:

$$
\begin{aligned}
P_A &: \; n = n + 1 \\
P_B &: \; n = n - 2 \\
P_C &: \; \text{print } n
\end{aligned}
$$

Η n έχει αρχική τιμή $n = 1$. Ζητείται να συγχρονίσετε κατάλληλα τις τρεις διεργασίες με χρήση της βιβλιοθήκης `pipesem` έτσι ώστε η έξοδος του προγράμματος να είναι `1 1 1 1 1 1 1 ...`

Η υλοποίησή σας θα πρέπει να ικανοποιεί τους εξής περιορισμούς:

- Όλες οι διεργασίες πρέπει να έχουν τη δυνατότητα να εκτελούνται – αποφυγή λιμοκτονίας
- Μπορείτε να δηλώσετε όσους σημαφόρους χρειάζεται, κατάλληλα αρχικοποιημένους. Στον κώδικα των διεργασιών, γράφετε μόνο στα σημεία που σημειώνονται με `/* ... */` στο αρχείο `procs-shm.c`.

Είναι εφικτή η υλοποίηση του σχήματος συγχρονισμού *μόνο* με κατάλληλα αρχικοποιημένους σημαφόρους και κλήσεις `pipesem_wait()` και `pipesem_signal()` στα σημεία που υποδηλώνονται;

Βήματα:

1. Αντιγράψτε το αρχείο `procs-shm.c` στο `ask3-3.c`, επεκτείνετε το `Makefile`. Αντιγράψτε τα `proc-common.{c,h}`, τα οποία έχουν επεκταθεί σε σχέση με την προηγούμενη άσκηση. Βεβαιωθείτε ότι το πρόγραμμα μεταγλωττίζεται χωρίς λάθη.
2. Εκτελέστε το πρόγραμμα ως έχει, χωρίς συγχρονισμό των διεργασιών. Παρατηρήστε ότι η P_C εκτυπώνει τιμές $n \neq 1$ και ανάλογο μήνυμα λάθους.
3. Υλοποιήστε κατάλληλο σχήμα συγχρονισμού ανάμεσα στις διεργασίες, εισάγοντας κλήσεις προς τη βιβλιοθήκη `pipesem`. **Προσοχή:** Δεν επιτρέπεται *καμία* αλλαγή στις `proc_a()`, `proc_b()`, `proc_c()` εκτός από εισαγωγή κώδικα πριν και μετά από το κρίσιμο τμήμα, μαζί με όσες δηλώσεις μεταβλητών ίσως χρειαστούν.
4. Βεβαιωθείτε ότι το νέο πρόγραμμα λειτουργεί σωστά, και η έξοδος περιέχει μόνο τιμές $n = 1$.

Ερωτήσεις:

1. Ποιος είναι ο σκοπός της κλήσης `create_shared_memory_area(sizeof(int))`, πριν από τη δημιουργία των τριών διεργασιών;
2. Πώς θα μπορούσε να γενικευτεί το σχήμα συγχρονισμού που υλοποιήσατε για την περίπτωση στην οποία η P_B εκτελούσε $n = n - K$, με K δεδομένη θετική σταθερά;

2 Εξέταση άσκησης και αναφορά

Η προθεσμία για την εξέταση της άσκησης στο εργαστήριο είναι η Πέμπτη 2012/01/19. Μετά την εξέταση η κάθε ομάδα θα πρέπει να συντάξει μια (σύντομη) αναφορά και να τη στείλει μέσω e-mail 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 διάφορα αρχεία εισόδου, ενδεικτικά.
- Σύντομες απαντήσεις στις ερωτήσεις.

3 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;