

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

**3η εργαστηριακή αναφορά**

στο μάθημα **“Λειτουργικά συστήματα”** 6ου Εξαμήνου

Από τους φοιτητές

Άγγελος Γκίκας , ΑΜ: 03118218

Νικήτας Τσίννας, ΑΜ: 03118187

Ομάδα: oslaba14

Προθεσμία παράδοσης: 25/5/2021

**Άσκηση 1: Συγχρονισμός σε υπάρχοντα κώδικα**

*Source code:*

Mε ‘Bold’ έχουν σημειωθεί οι γραμμές κώδικα που άλλαξαν/προστέθηκαν.

*/\**

*\* simplesync.c*

*\**

*\* A simple synchronization exercise.*

*\**

*\* Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>*

*\* Operating Systems course, ECE, NTUA*

*\**

*\*/*

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

*/\**

*\* POSIX thread functions do not return error numbers in errno,*

*\* but in the actual return value of the function call instead.*

*\* This macro helps with error reporting in this case.*

*\*/*

#define perror\_pthread(ret, msg) \

do { errno = ret; perror(msg); } while (0)

#define N 10000000

*/\* Dots indicate lines where you are free to insert code at will \*/*

*/\* ... \*/*

#if defined(SYNC\_ATOMIC) ^ defined(SYNC\_MUTEX) == 0

# error You must #define exactly one of SYNC\_ATOMIC or SYNC\_MUTEX.

#endif

#if defined(SYNC\_ATOMIC)

# define USE\_ATOMIC\_OPS 1

#else

# define USE\_ATOMIC\_OPS 0

#endif

pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER;

void \*increase\_fn(void \*arg)

{

int i;

**volatile** int \*ip = arg;

fprintf(stderr, "About to increase variable %d times**\n**", N);

**for** (i = 0; i < N; i++) {

**if** (USE\_ATOMIC\_OPS) {

*/\* ... \*/*

*/\* You can modify the following line \*/*

*//++(\*ip);*

**\_\_sync\_fetch\_and\_add(ip,1);**

*/\* ... \*/*

} **else** {

**pthread\_mutex\_lock(&lock);**

*/\* ... \*/*

*/\* You cannot modify the following line \*/*

++(\*ip);

*/\* ... \*/*

**pthread\_mutex\_unlock(&lock);**

}

}

fprintf(stderr, "Done increasing variable.**\n**");

**return** NULL;

}

void \*decrease\_fn(void \*arg)

{

int i;

**volatile** int \*ip = arg;

fprintf(stderr, "About to decrease variable %d times**\n**", N);

**for** (i = 0; i < N; i++) {

**if** (USE\_ATOMIC\_OPS) {

*/\* ... \*/*

*/\* You can modify the following line \*/*

*//--(\*ip);*

**\_\_sync\_fetch\_and\_sub(ip,1);**

*/\* ... \*/*

} **else** {

**pthread\_mutex\_lock(&lock);**

*/\* ... \*/*

*/\* You cannot modify the following line \*/*

--(\*ip);

*/\* ... \*/*

**pthread\_mutex\_unlock(&lock);**

}

}

fprintf(stderr, "Done decreasing variable.**\n**");

**return** NULL;

}

int main(int argc, char \*argv[])

{

int val, ret, ok;

pthread\_t t1, t2;

*/\**

*\* Initial value*

*\*/*

val = 0;

*/\**

*\* Create threads*

*\*/*

ret = pthread\_create(&t1, NULL, increase\_fn, &val);

**if** (ret) {

perror\_pthread(ret, "pthread\_create");

exit(1);

}

ret = pthread\_create(&t2, NULL, decrease\_fn, &val);

**if** (ret) {

perror\_pthread(ret, "pthread\_create");

exit(1);

}

*/\**

*\* Wait for threads to terminate*

*\*/*

ret = pthread\_join(t1, NULL);

**if** (ret)

perror\_pthread(ret, "pthread\_join");

ret = pthread\_join(t2, NULL);

**if** (ret)

perror\_pthread(ret, "pthread\_join");

*/\**

*\* Is everything OK?*

*\*/*

ok = (val == 0);

printf("%sOK, val = %d.**\n**", ok ? "" : "NOT ", val);

**return** ok;

}

*Ενδεικτικοί έξοδοι εκτέλεσης προγράμματος μετά την επεξεργασία αρχείου κώδικα:*

* **oslaba14@orion:~/ex3\_1$ time ./simplesync-atomic**

About to decrease variable 10000000 times

About to increase variable 10000000 times

Done increasing variable.

Done decreasing variable.

OK, val = 0.

real 0m1.125s

user 0m2.232s

sys 0m0.000s

* **oslaba14@orion:~/ex3\_1$ time ./simplesync-mutex**

About to decrease variable 10000000 times

About to increase variable 10000000 times

Done decreasing variable.

Done increasing variable.

OK, val = 0.

real 0m4.223s

user 0m4.112s

sys 0m3.756s

*Ερωτήσεις*

**1.**

| Χωρίς Συγχρονισμό | POSIX-MUTEX | GCC-ATOMIC |
| --- | --- | --- |
| 0.345 s | 0.4223 s | 1.125 s |

Συγκρίνοντας τα παραπάνω αποτελέσματα συμπεραίνουμε πως η παράλειψη συγχρονισμού των νημάτων είναι ταχύτερη σε σχέση με τον συγχρονισμό αυτών. Αυτό άλλωστε φαίνεται λογικό εφόσον ο συγχρονισμός, μπορεί να είναι απαραίτητος, αλλά απαιτεί από την άλλη αναμονή των νημάτων μεταξύ τους για την ορθή εκτέλεση του προγράμματος.

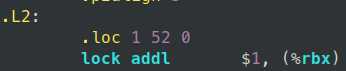
**2.**

Η μέθοδος με χρήση ατομικών λειτουργιών είναι γρηγορότερη, καθώς υλοποιείται σε low level, σε αντίθεση με τη χρήση POSIX mutexes, και άρα απαιτείται μικρότερος αριθμός εντολών σε γλώσσα μηχανής. Οι εντολές που έχουν χρησιμοποιηθεί (\_\_sync\_fetch\_and\_add(ip,1) και \_\_sync\_fetch\_and\_sub(ip,1)) φροντίζουν να αποθηκεύουν την νέα τιμή στην αντίστοιχη θέση μνήμης πριν από οποιοδήποτε άλλο πρόγραμμα αποκτήσει πρόσβαση σε αυτήν.

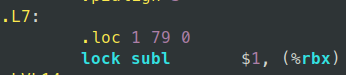
Από την άλλη, η μέθοδος με την χρήση mutexes λειτουργεί με κλείδωμα κρίσιμων κομματιών κώδικα ώστε να μην υπάρχει πρόσβαση από κάποιο ξένο thread σε αυτό. Εδώ, το κρίσιμο κομμάτι είναι η ανάθεση της νέας τιμής στην μεταβλητή val. Αυτή η τεχνική είναι υψηλότερου επιπέδου και μεταφράζεται σε περισσότερες εντολές σε γλώσσα assembly από την ατομική μέθοδο, όπως φαίνεται και σε επόμενη ερώτηση.

**3.**

Παρατηρούμε πως η εντολή \_\_sync\_fetch\_and\_add(ip,1) της γραμμής 52 μεταφράζεται σε μία εντολή assembly:

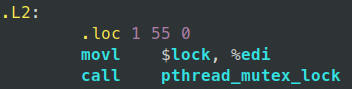


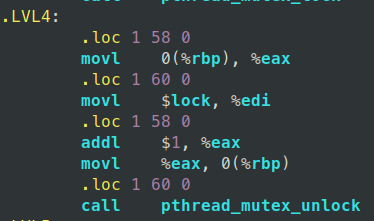
Αντίστοιχα και η \_\_sync\_fetch\_and\_sub(ip,1) της γραμμής 79:



**4.**

H εντόλή για το κλείδωμα (.loc 1 55 0) και το ξεκλείδωμα (.loc 1 60 0)του mutex αντίστοιχα μεταφράζονται στις ακόλουθες εντολές assembly:





Παρατηρούμε πως οι εντολές στην γλώσσα assembly για την υλοποίηση του συγχρονισμού με mutexes είναι περισσότερες σε σχέση με την μία εντολή της υλοποίησης με των ατομικών λειτουργιών GCC.

**Άσκηση 2: Δημιουργία αυθαίρετου δέντρου διεργασιών**

*Source code:*

*/\**

*\* mandel.c*

*\**

*\* A program to draw the Mandelbrot Set on a 256-color xterm.*

*\**

*\*/*

#include <stdio.h>

#include <unistd.h>

#include <assert.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

#include <semaphore.h>

#include <pthread.h>

#include <signal.h>

#include "mandel-lib.h"

#define MANDEL\_MAX\_ITERATION 100000

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Compile-time parameters \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**typedef** **struct** *//new: thread struct*

{

pthread\_t tid;

int current\_line;

sem\_t mutex;

}new\_thread;

int threads; *//new: total threads used*

new\_thread \*thread\_ptr; *//new: array of new\_thread (s)*

*/\**

*\* Output at the terminal is is x\_chars wide by y\_chars long*

*\*/*

int y\_chars = 50;

int x\_chars = 90;

*/\**

*\* The part of the complex plane to be drawn:*

*\* upper left corner is (xmin, ymax), lower right corner is (xmax, ymin)*

*\*/*

double xmin = -1.8, xmax = 1.0;

double ymin = -1.0, ymax = 1.0;

*/\**

*\* Every character in the final output is*

*\* xstep x ystep units wide on the complex plane.*

*\*/*

double xstep;

double ystep;

*/\**

*\* This function computes a line of output*

*\* as an array of x\_char color values.*

*\*/*

void compute\_mandel\_line(int line, int color\_val[])

{

*/\**

*\* x and y traverse the complex plane.*

*\*/*

double x, y;

int n;

int val;

*/\* Find out the y value corresponding to this line \*/*

y = ymax - ystep \* line;

*/\* and iterate for all points on this line \*/*

**for** (x = xmin, n = 0; n < x\_chars; x+= xstep, n++) {

*/\* Compute the point's color value \*/*

val = mandel\_iterations\_at\_point(x, y, MANDEL\_MAX\_ITERATION);

**if** (val > 255)

val = 255;

*/\* And store it in the color\_val[] array \*/*

val = xterm\_color(val);

color\_val[n] = val;

}

}

*/\**

*\* This function outputs an array of x\_char color values*

*\* to a 256-color xterm.*

*\*/*

void output\_mandel\_line(int fd, int color\_val[])

{

int i;

char point ='@';

char newline='\n';

**for** (i = 0; i < x\_chars; i++) {

*/\* Set the current color, then output the point \*/*

set\_xterm\_color(fd, color\_val[i]);

**if** (write(fd, &point, 1) != 1) {

perror("compute\_and\_output\_mandel\_line: write point");

exit(1);

}

}

*/\* Now that the line is done, output a newline character \*/*

**if** (write(fd, &newline, 1) != 1) {

perror("compute\_and\_output\_mandel\_line: write newline");

exit(1);

}

}

void\* compute\_and\_output\_mandel\_line(void\* arg) *//changed*

{

int line = \*(int\*) arg;

*/\**

*\* A temporary array, used to hold color values for the line being drawn*

*\*/*

int color\_val[x\_chars];

int i;

**for**(i = line; i < y\_chars; i += threads)

{

compute\_mandel\_line(i, color\_val);

sem\_wait(&thread\_ptr[(i % threads)].mutex);

output\_mandel\_line(STDOUT\_FILENO, color\_val);

sem\_post(&thread\_ptr[((i % threads)+1) % threads].mutex);

}

**return** NULL;

}

*//function used in signal(SIGINT, sighandler) in main()*

void sighandler()

{

reset\_xterm\_color(1); *//reset terminal color*

printf("%s", "**\n**"); *//change line in terminal*

exit(1); *//exit program*

}

int main(int argc, char\* argv[])

{

*//if termination signal is detected go to sighandler*

signal(SIGINT, sighandler);

int line;

threads = atoi(argv[1]); *//new: read number of threads*

xstep = (xmax - xmin) / x\_chars;

ystep = (ymax - ymin) / y\_chars;

thread\_ptr = (new\_thread \*)malloc(threads \* **sizeof**(new\_thread)); *//new: make space in heap*

**if** (thread\_ptr == NULL)

{

fprintf(stderr, "Not enough memory for allocation.**\n**");

exit(1);

}

**if** ((sem\_init(&thread\_ptr[0].mutex, 0, 1)) == -1) *//new: intialize first lock*

{

fprintf(stderr, "Semaphore.**\n**");

exit(1);

}

**for** (line = 1; line < threads; line++) *//new: initialize "locks"*

{

**if** ((sem\_init(&thread\_ptr[line].mutex, 0, 0)) == -1)

{

fprintf(stderr, "Semaphore.**\n**");

exit(1);

}

}

**for** (line = 0; line < threads; line++) *//new: create threads*

{

thread\_ptr[line].current\_line = line;

**if** ((pthread\_create(&thread\_ptr[line].tid, NULL,

compute\_and\_output\_mandel\_line, &thread\_ptr[line].current\_line)) != 0)

{

*/\* The pthread\_create function returns 0 on success \*/*

fprintf(stderr, "Error at thread(s) creation.**\n**");

exit(1);

}

}

**for** (line = 0; line < threads; line++) *//new: termination of threads*

{

**if** ((pthread\_join(thread\_ptr[line].tid, NULL)) != 0)

{

*/\* As pthread\_create, pthread\_join function returns 0 on success \*/*

fprintf(stderr, "Error at thread(s) termination.**\n**");

exit(1);

}

}

**for** (line = 0; line < threads; line++) *//new: destroy semaphores*

{

sem\_destroy(&thread\_ptr[line].mutex);

}

free(thread\_ptr);

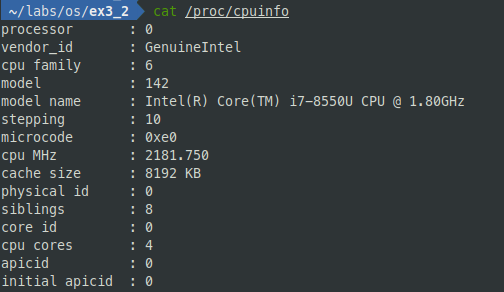
reset\_xterm\_color(1);

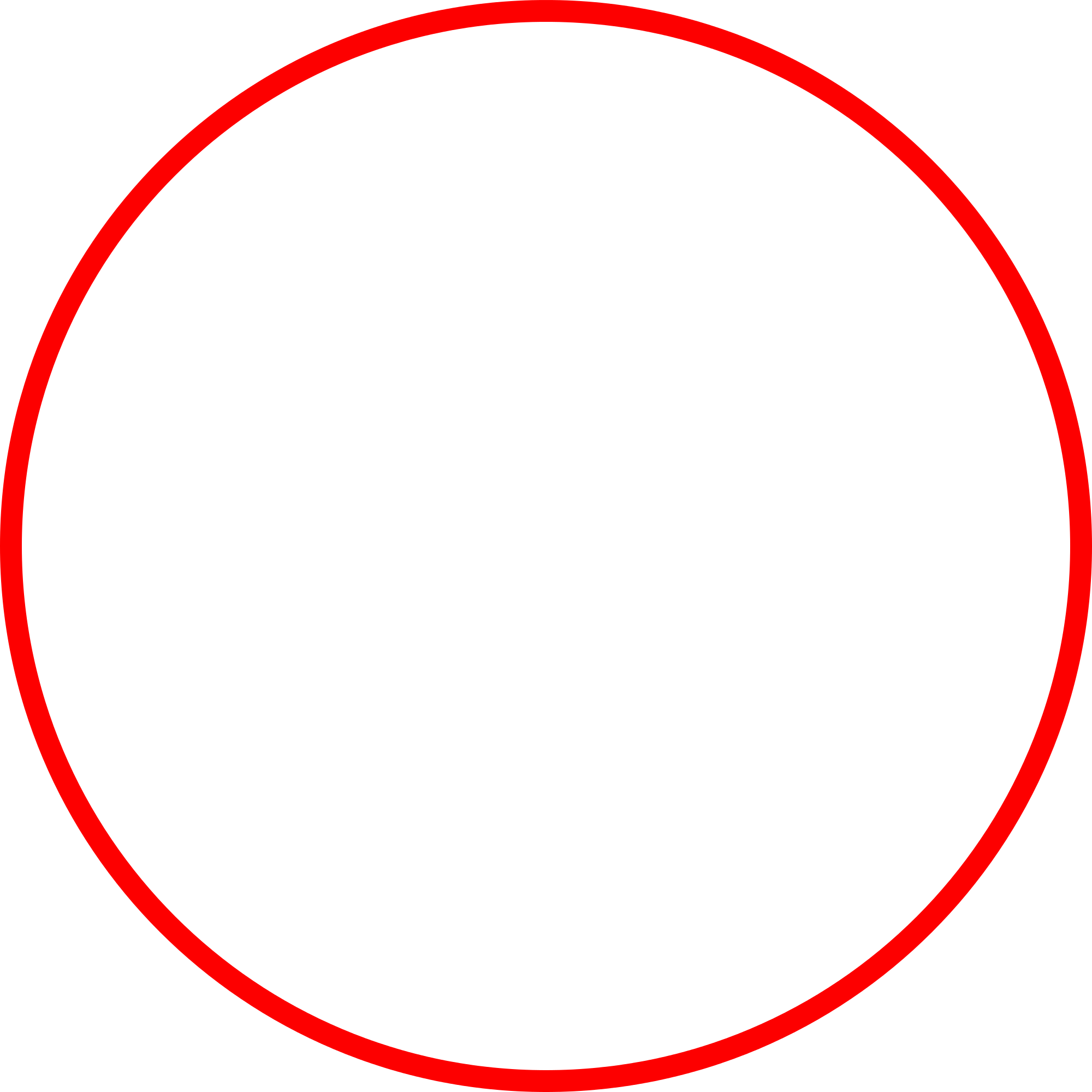
**return** 0;

}

*Ενδεικτικοί έξοδοι εκτέλεσης προγράμματος μετά την επεξεργασία αρχείου κώδικα:*

Αρχικά, επισημαίνουμε πως το πρόγραμμα το εκτελέσαμε σε μηχανή με επεξεργαστή 4 πυρήνων όπως φαίνεται στο παρακάτω screenshot:

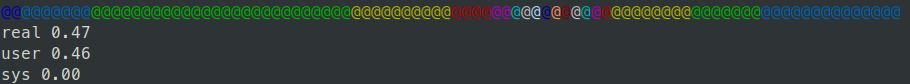




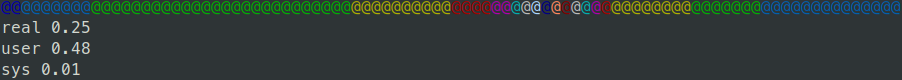
Πρώτα εκτελούμε το αρχικό πρόγραμμα σειριακού υπολογισμού και έπειτα το αντίστοιχο παράλληλου υπολογισμού χρησιμοποιώντας 2 threads. Εφόσον οι πυρήνες είναι περισσότεροι από τα νήματα που χρησιμοποιούμε, ο παράλληλος υπολογισμός έχει νόημα.



Χρόνος εκτέλεσης χωρίς χρήση νημάτων:



Χρόνος εκτέλεσης με χρήση 2 νημάτων:



*Ερωτήσεις*

**1.**

Χρειάζονται τόσοι σημαφόροι όσα είναι και τα νήματα τα οποία επιθυμούμε να τρέχουν ταυτόχρονα, καθώς κάθε σημαφόρος είναι υπεύθυνος για το συγχρονισμό ενός νήματος. Πιο συγκεκριμένα, αρχικοποιούμε πρώτα τους σημαφόρους στο 0 εκτός από τον πρώτο στον πίνακα. Αρχικά, το κάθε νήμα αρχίζει να υπολογίζει την γραμμή που του αντιστοιχεί. Να σημειωθεί εδώ πως αν τα νήματα είναι λιγότερα από τις γραμμές εκτύπωσης τότε το κάθε νήμα θα αναλαμβάνει την αντίστοιχη γραμμή που του αναλογεί κυκλικά. Έπειτα, αφού γίνει ο υπολογισμός του πρώτου νήματος, αυτό θα εκτυπώσει την γραμμή του στο τερματικό και έπειτα θα δώσει εντολή στον επόμενο σημαφόρο να αυξήσει την τιμή του κατά 1. Με αυτόν τον τρόπο το επόμενο νήμα θα ξεκινήσει την εκτύπωση, ενώ θα έχει εκτελέσει ήδη τους υπολογισμούς. Αυτή η διαδικασία θα συνεχιστεί παρομοίως μέχρι να εκτυπωθεί το τελικό σχήμα.

**2.**

|  | Mandel - no threads | Mandel - 2 threads |
| --- | --- | --- |
| real | 0.47 s | 0.25 s |
| user | 0.46 s | 0.48 s |
| system | 0.00 s | 0.01 s |

Παρατηρούμε πως ο συνολικός χρόνο εκτέλεσης έχει μειωθεί σε μεγάλο βαθμό στην περίπτωση των 2 νημάτων όπως άλλωστε ήταν αναμενόμενο. Αυτό οφείλεται στο μοίρασμα υπολογιστικού φόρτου που επιτυγχάνεται μέσω των 2 νημάτων σε ξεχωριστούς πυρήνες επεξεργαστή του μηχανήματος.

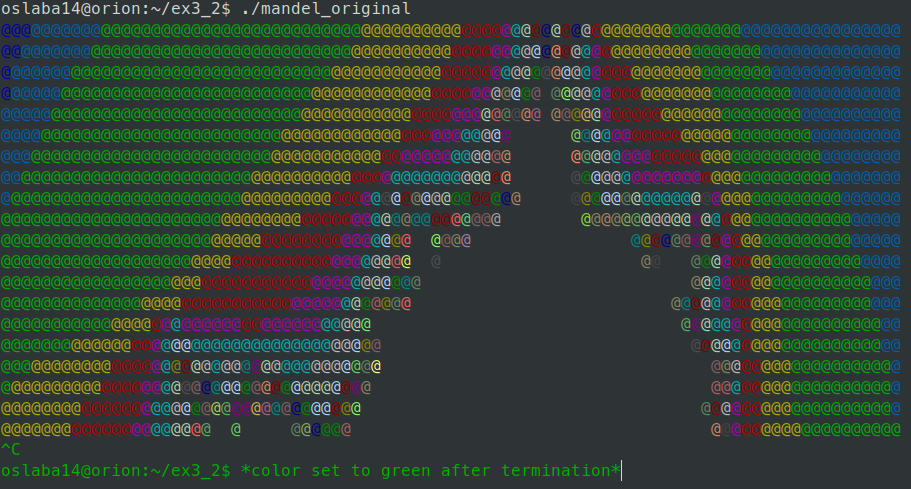
**3.**

Το πρόγραμμα που έχουμε υλοποιήσει φαίνεται να εμφανίζει σημαντική επιτάχυνση συγκριτικά με το πρώτο εκτελέσιμο mandel. Αυτό, όπως ειπώθηκε και προηγουμένως, οφείλεται στο ότι τα νήματα εκτελούν τους απαραίτητους υπολογισμούς κατάλληλα και απαιτούν συγχρονισμό μόνο στην διαδικασία του τυπώματος των γραμμών. Δηλαδή, κάθε νήμα δεν περιμένει το προηγούμενο να τελειώσει για να υπολογίσει την γραμμή που του αντιστοιχεί, αλλά την έχει έτοιμη για εκτύπωση όταν τερματίσει την διαδικασία του τυπώματος το προηγούμενο από αυτό νήμα.

Επομένως, στο κρίσιμο τμήμα του προγράμματος, που αφορά τον συγχρονισμό νημάτων, περιέχεται μόνο το κομμάτι του τυπώματος και όχι του υπολογισμού. Αν, ωστόσο, το είχαμε συμπεριλάβει, τότε θα ήταν λογικό να μην εμφανιζόταν κάποια επιτάχυνση, εφόσον η διαδικασία θα ήταν η ίδια με το πρώτο εκτελέσιμο mandel με την μόνη διαφορά πως θα αναλάμβανε συνολικά κάθε νήμα μία γραμμή σειριακά.

**4.**

Αν τερματίσουμε το πρόγραμμα νωρίτερα, το χρώμα των χαρακτήρων του τερματικού θα γινόταν ίδιο με τον τελευταίο χαρακτήρα που τυπώθηκε κατά την εκτέλεση. Αυτό συμβαίνει γιατί ο τρόπος που έχουμε υλοποιήσει το τύπωμα των χαρακτήρων με διαφορετικό χρώμα είναι μέσω αλλαγής του προκαθορισμένου χρώματος χαρακτήρων του τερματικού:



Για να το διορθώσουμε αυτό μπορούμε να χρησιμοποιήσουμε μία ρουτίνα sighandler εφόσον το CTRL+C του χρήστη αντιστοιχεί στο σήμα SIGINT (interrupt signal). Έτσι, όπως φαίνεται και στο παρακάτω printout του κώδικα, έχουμε υλοποιήσει την συνάρτηση sighandler στην οποία επαναφέρουμε το χρώμα του τερματικού στο αρχικό, αλλάζουμε γραμμή και έπειτα τερματίζουμε το πρόγραμμα.

