ΕΘΝΙΚΟ ΜΕΤΣΟΒΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ: ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΑΘΗΜΑ: Λειτουργικά Συστήματα

3η Εργαστηριακή Άσκηση

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΜΕΛΩΝ :

ΣΤΑΜΑΤΗΣ ΑΛΕΞΑΝΔΡΟΠΟΥΛΟΣ (03117060)

ΠΟΛΥΤΙΜΗ-ΑΝΝΑ ΓΚΟΤΣΗ (03117201)

ΑΘΗΝΑ

2020

Στόχος της 3ης εργαστηριακής άσκησης ήταν η χρήση διαδεδομένων μηχανισμών συγχρονισμού για την επίλυση προβλημάτων συγχρονισμού σε πολυνηματικές εφαρμογές, οι οποίες βασίζονται στο πρότυπο POSIX threads. Οι μηχανισμοί αυτοί που χρησιμοποιήσαμε είναι:
α)
•κλειδώματα
•σημαφόροι
•μεταβλητές συνθήκης
β)
•ατομικές λειτουργίες (όπως ορίζονται από το υλικό και εξάγονται στον προγραμματιστή μέσω ειδικών εντολών (builtins) του μεταγλωττιστή GCC

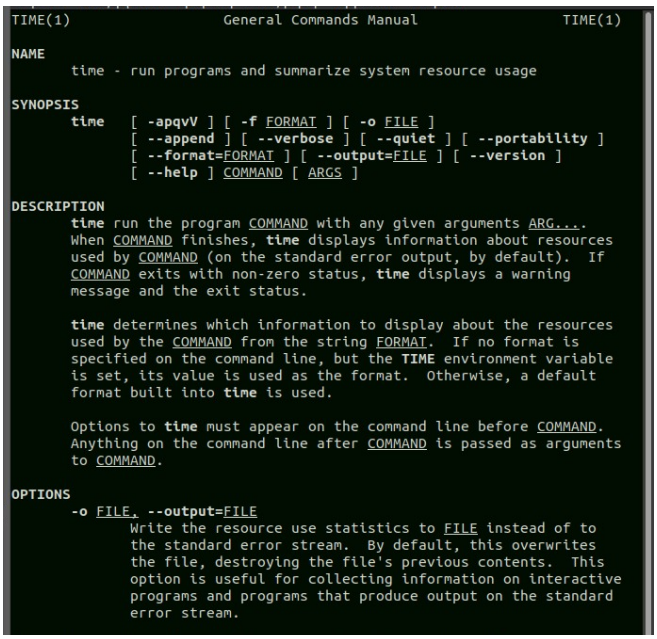
Άσκηση 1η

Πρωταρχικός στόχος της άσκησης αυτής είναι ο συγχρονισμός δυό νημάτων εκτέλεσης, ένος που αυξάνει Ν φορές τη τιμή μιας μοιραζόμενης μεταβλητής και ενός που την μειώνει, ώστε μετά το πέρας της εκτέλεσης των νημάτων να αποκτήσουμε τα σωστά αποτελέσματα. Αρχικά χρησιμοποιήσαμε το δοθέν Makefile προκειμένου να μεταγλωττίσουμε και να τρέξουμε το πρόγραμμα μας. Παρατηρούμε ότι κατά την εκτέλεση παράγονται δύο εκτελέσιμα, το simplesync-atomic και simplesync-mutex. Αυτό γίνεται μέσω των #define που υπάρχουν μέσα στον κώδικα. Όπως φαίνεται και από το Makefile μέσω των εντολών :

```
simplesync-mutex.o: simplesync.c
	$(CC) $(CFLAGS) -DSYNC_MUTEX -c -o simplesync-mutex.o simplesync.c

simplesync-atomic.o: simplesync.c
	$(CC) $(CFLAGS) -DSYNC_ATOMIC -c -o simplesync-atomic.o simplesync.c
```

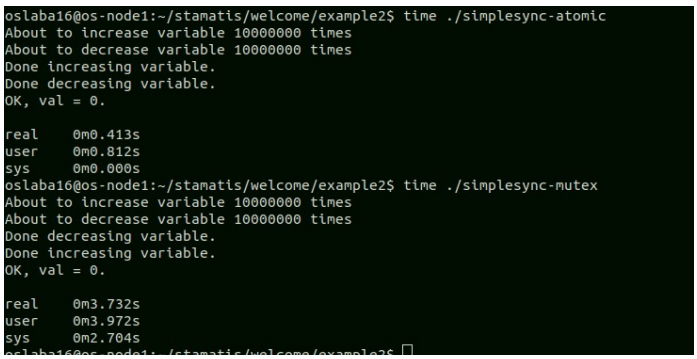
κατά το compilation, για την δημιουργία του object file simplesync-mutex.o (και κατ' επέκταση του εκτελέσιμου simplesync-mutex) περνάμε την παράμετρο DSYNC_MUTEX, η οποία περιλαμβάνει κομμάτια κώδικα που βρίσκονται μέσα στα τμήματα # if define SYNC_MUTEX, ενώ για την δημιουργία του object file simplesync-atomic.o (και κατ' επέκταση του εκτελέσιμου simplesync-atomic) περνάμε την παράμετρο DSYNC_ATOMIC, η οποία περιλαμβάνει κομμάτια κώδικα που βρίσκονται μέσα στα τμήματα # if define SYNC_ATOMIC.

1.Αρχικά έγινε χρήση της εντολής time(1), αφού πρώτα τη μελετήσαμε τη λειτουργία της από το manual pages

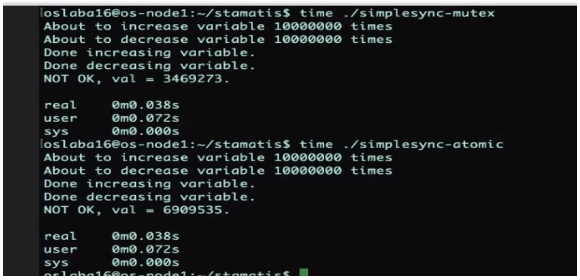


```
TIME(1)                  General Commands Manual                 TIME(1)

NAME
       time - run programs and summarize system resource usage

SYNOPSIS
       time   [ -apqvV ] [ -f FORMAT ] [ -o FILE ]
              [ --append ] [ --verbose ] [ --quiet ] [ --portability ]
              [ --format=FORMAT ] [ --output=FILE ] [ --version ]
              [ --help ] COMMAND [ ARGS ]

DESCRIPTION
       time run the program COMMAND with any given arguments ARG....
       When COMMAND finishes, time displays information about resources
       used by COMMAND (on the standard error output, by default).  If
       COMMAND exits with non-zero status, time displays a warning
       message and the exit status.

       time determines which information to display about the resources
       used by the COMMAND from the string FORMAT.  If no format is
       specified on the command line, but the TIME environment variable
       is set, its value is used as the format.  Otherwise, a default
       format built into time is used.

       Options to time must appear on the command line before COMMAND.
       Anything on the command line after COMMAND is passed as arguments
       to COMMAND.

OPTIONS
       -o FILE, --output=FILE
              Write the resource use statistics to FILE instead of to
              the standard error stream.  By default, this overwrites
              the file, destroying the file's previous contents.  This
              option is useful for collecting information on interactive
              programs and programs that produce output on the standard
              error stream.
```

Έχουμε:
Για τον χρόνο εκτέλεσης και την έξοδο των εκτελέσιμων που εκτελούν συγχρονισμό έχουμε:



Για το χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό έχουμε:



Με βάση τα παραπάνω είναι προφανές ότι ο χρόνος εκτέλεσης αυξάνεται αφού υλοποιούμε σχήμα συγχρονισμού στην ενημέρωση των τιμών της κοινής μεταβλητής. Βλέπουμε δηλαδή ότι γρηγορότερο είναι το εκτελέσιμο χωρίς συγχρονισμό, έπεται το simplesync-atomic και τέλος το simplesync-mutex.
Ο λόγος που ο χρόνος εκτέλεσης του προγράμματος χωρίς συγχρονισμό είναι μικρότερος είναι ότι σε αυτό δεν υπάρχει κάποιος έλεγχος-περιορισμός για την είσοδο των threads στο κρίσιμο τμήμα, και άρα αυτά δεν χρειάζεται ποτέ να περιμένουν για να αυξήσουν ή να μειώσουν την μεταβλητή, αναμονή η οποία προφανώς προκαλεί καθυστερήσεις.

2. Η χρήση των ατομικών λειτουργιών είναι γρηγορότερη. Αυτό φαίνεται, αφού το simplesync-atomic που υλοποιείται με atomic operations, είναι πιο γρήγορο από το simplesync-mutex που χρησιμοποιεί pthread mutexes. Τα atomic operations υποστηρίζονται συνήθως από τον επεξεργαστή, ενώ τα mutexes είναι πιο high level υλοποιήσεις και απαιτούν τουλάχιστον δύο atomic operations, μία για το lock και ένα για το unlock. Επιπλέον, στα mutexes, στην περίπτωση που ένα thread επιχειρήσει να εκτελέσει ένα κομμάτι κώδικα που είναι κλειδωμένο από ένα άλλο thread, αναστέλλει συνήθως την λειτουργία του, απελευθερώνοντας τον επεξεργαστή για άλλα threads, και επομένως είναι πιο αργό στο να επανέλθει μετά το unlock. Κάτι τέτοιο δεν συμβαίνει όταν χρησιμοποιούμε atomic operations, κατά τα οποία τα threads συνεχίζουν να προσπαθούν μέχρι να τους επιτραπεί να συνεχίσουν στο κρίσιμο τμήμα, χωρίς να μπλοκαριστούν ποτέ. Τα παραπάνω εξηγούν επομένως, γιατί παρατηρείται μικρότερος χρόνος στην υλοποίηση με atomic operations.

3)Για το σκοπό του ερωτήματος αυτού τροποποιήσαμε το Makefile εισάγωντας την εντολή:

simplesync-atomic_asembly: simplesync.c
$(CC) $(CFLAGS) -DSYNC_ATOMIC simplesync.c -S -g -o simplesync-atomic.S

Έτσι, ύστερα από χρήση της εντολής make και κατ'επέκταση μετά από εκτέλεση αυτής της εντολής, ο ενδιάμεσος κώδικας assembly θα βρίσκεται στο αρχείο simplesync-atomic.S. Με την εντολή cat simplesync-atomic.S ανοίγουμε το αρχείο αυτό και βλέπουμε ότι τα κομμάτια assembly που αντιστοιχούν στη μετάφραση της χρήσης ατομικών λειτουργιών είναι τα εξής:

```
.L2:
        .loc 1 50 0
        lock addl       $1, (%rbx)
```

```
.L7:
        .loc 1 77 0
        lock subl       $1, (%rbx)
```

4) Εντελώς αντίστοιχα με το παραπάνω ερώτημα τροποποιήσαμε το Makefile εισάγωντας την εντολή:
simplesync-mutex_asembly: simplesync.c
$(CC) $(CFLAGS) -DSYNC_MUTEX simplesync.c -S -g -o simplesync-mutex.S

Έτσι, ύστερα από χρήση της εντολής make και κατ'επέκταση μετά από εκτέλεση αυτής της εντολής, ο ενδιάμεσος κώδικας assembly θα βρίσκεται στο αρχείο simplesync-mutex.S. Με την εντολή cat simplesync-mutex.S ανοίγουμε το αρχείο αυτό και βλέπουμε ότι τα κομμάτια assembly που παράγονται από την pthread_mutex_lock() είναι τα ακόλουθα:

```
.L2:
        .loc 1 53 0
        movl    $mutex, %edi
        call    pthread_mutex_lock
.LVL4:
        .loc 1 56 0
        movl    0(%rbp), %eax
        .loc 1 58 0
        movl    $mutex, %edi
        .loc 1 56 0
        addl    $1, %eax
        movl    %eax, 0(%rbp)
        .loc 1 58 0
        call    pthread_mutex_unlock
```

Παρακάτω επισυνάπτεται τμήμα του τροποποιημένο Makefile για τα ερωτήματα 3,4, όπου φαίνεται ότι έχουμε εισάγει τις εντολές :

simplesync-mutex_asembly: simplesync.c
$(CC) $(CFLAGS) -DSYNC_MUTEX simplesync.c -S -g -o simplesync-mutex.S

simplesync-atomic_asembly: simplesync.c
$(CC) $(CFLAGS) -DSYNC_ATOMIC simplesync.c -S -g -o simplesync-atomic.S

```
CFLAGS = -Wall -O2 -pthread
LIBS =

all: pthread-test simplesync-mutex simplesync-atomic simplesync-mutex_asembly simplesync-atomic_asembly kgarten mandel

## Pthread test
pthread-test: pthread-test.o
	$(CC) $(CFLAGS) -o pthread-test pthread-test.o $(LIBS)

pthread-test.o: pthread-test.c
	$(CC) $(CFLAGS) -c -o pthread-test.o pthread-test.c

## Simple sync (two versions)
simplesync-mutex: simplesync-mutex.o
	$(CC) $(CFLAGS) -o simplesync-mutex simplesync-mutex.o $(LIBS)

simplesync-atomic: simplesync-atomic.o
	$(CC) $(CFLAGS) -o simplesync-atomic simplesync-atomic.o $(LIBS)

simplesync-mutex.o: simplesync.c
	$(CC) $(CFLAGS) -DSYNC_MUTEX -c -o simplesync-mutex.o simplesync.c

simplesync-atomic.o: simplesync.c
	$(CC) $(CFLAGS) -DSYNC_ATOMIC -c -o simplesync-atomic.o simplesync.c

simplesync-mutex_asembly: simplesync.c
	$(CC) $(CFLAGS)  -DSYNC_MUTEX simplesync.c -S -g -o simplesync-mutex.S

simplesync-atomic_asembly: simplesync.c
	$(CC) $(CFLAGS)  -DSYNC_ATOMIC simplesync.c -S -g -o simplesync-atomic.S

## Kindergarten
kgarten: kgarten.o
	$(CC) $(CFLAGS) -o kgarten kgarten.o $(LIBS)
```

Επίσης παρατίθεται ο κώδικας της Άσκησης 1 που υπάρχει στο αρχείο /home/oslab/oslaba16/exc3/ simplesync.c

```
/*
 * 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 mutex= 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(&mutex);
			/* ... */
			/* You cannot modify the following line */
			++(*ip);
			/* ... */
			pthread_mutex_unlock(&mutex);
		}
	}
	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(&mutex);
			/* ... */
```

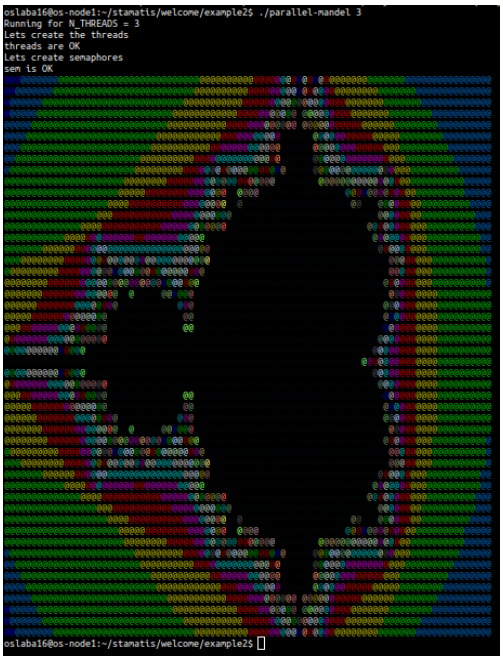
```
                /* You cannot modify the following line */
                --(*ip);
                pthread_mutex_unlock(&mutex);
                /* ... */
        }
    }
    fprintf(stderr, "Done decreasing variable.\n");

    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;
}
```

Άσκηση 2η
Παράλληλος υπολογισμός του συνόλου Mandelbrot

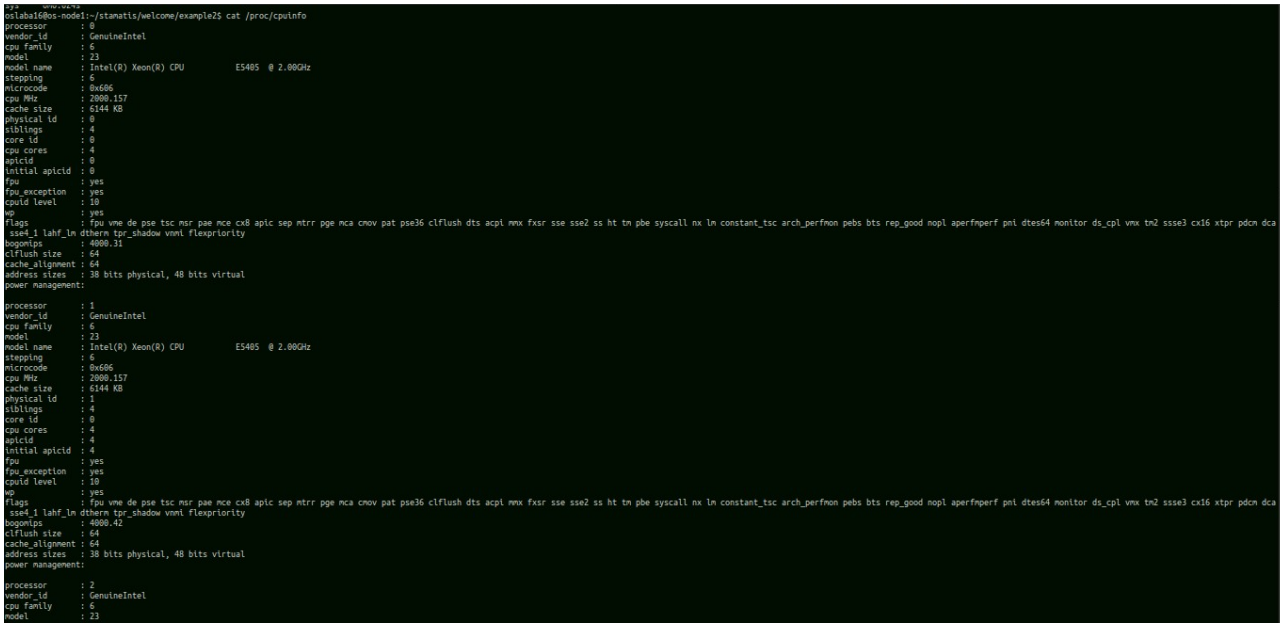
Το πρόγραμμα που υπάρχει στο mandel.c υπολογίζει και σχεδιάζει το σύνολο Mandelbrot σε τερματικό κειμένου, χρησιμοποιώντας χρωματιστούς χαρακτήρες. Για τη λειτουργία του βασίζεται στη βιβλιοθήκη mandel-lib.{c, h}.Η έξοδος του προγράμματος mandel είναι ένα μπλοκ χαρακτήρων, διαστάσεων x_chars στηλών και y_chars γραμμών, όπως φαίνεται παρακάτω για NTHREADS=3:

Για το parallel-mandel, που υλοποιήσαμε, του οποίου ο κώδικας φαίνεται παρακάτω η έξοδος είναι η ίδια με την παραπάνω, όπως φαίνεται παρακάτω:

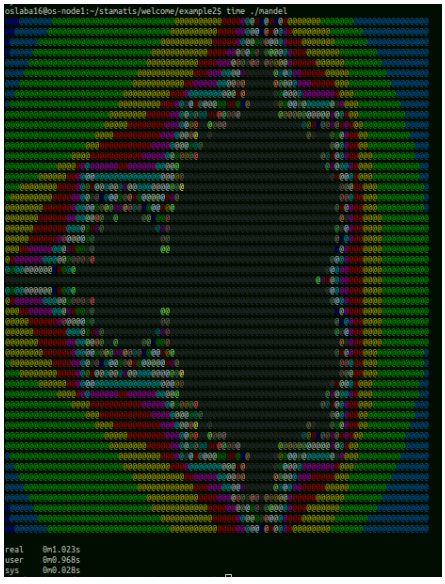


1) Το πλήθος των σημαφόρων που απαιτούνται είναι ίδιο με το πλήθος των threads που δημιουργούμε για την υλοποίηση μας στο αρχείο parallel-mandel.c και άρα απαιτούνται Ν σημαφόροι. (Οι σημαφόροι χρησιμοποιούνται για τον συγχρονισμό του τυπώματος των υπολογισμένων γραμμών, κάθε thread για να τυπώσει ελέγχει έναν σημαφόρο, και στην συνέχεια, όταν τυπώσει την γραμμή που είχε υπολογίσει κάνει sem_post στον σημαφόρο που είναι επόμενος, ώστε να τυπωθεί η επόμενη γραμμή με το επόμενο thread).

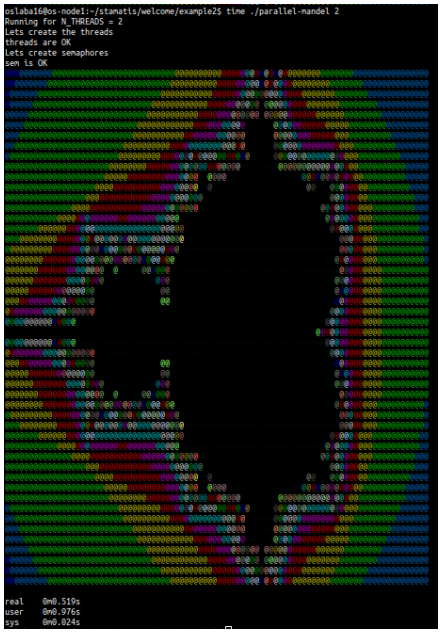
2) Εκτελώντας την εντολή ***"cat proc/cpuinfo"*** μπορούμε να βρούμε πληροφορίες σχετικά με το σύστημα μας. Συγκεκριμένα βλέπουμε τα εξής:



Από τα παραπάνω φαίνεται ότι το σύστημα μας είνια 4πύρηνο αφού cpu cores : 4
Για τον κώδικα με την σειριακή υλοποίηση παίρνουμε:

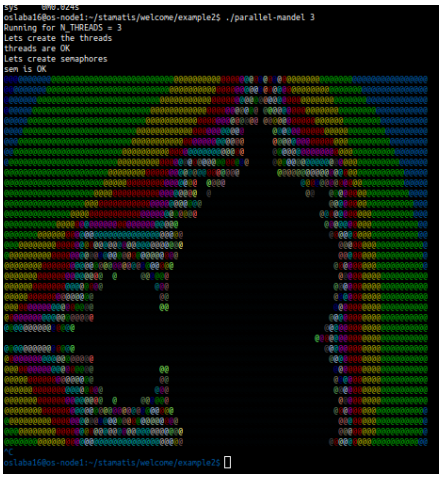


Για τον κώδικα με την παράλληλη υλοποίηση με 2 νήματα, όπως φαίνεται από την εντολή time ./parallel-mandel 2 παίρνουμε:



3) Παρατηρούμε από τις παραπάνω μετρήσεις ότι το πρόγραμμα με τον παράλληλο υπολογισμό του Mandelbrot με threads παρουσιάζει επιτάχυνση σε σχέση με το σειριακό. Κάτι τέτοιο, είναι λογικό, καθώς η φάση υπολογισμού γίνεται παράλληλα και ο συγχρονισμός χρησιμοποιείται ώστε να γίνει με τη κατάλληλη σειρά το τύπωμα των χαρακτήρων. Το κρίσιμο τμήμα επομένως περιέχει μόνο την φάση εξόδου κάθε παραγόμενης γραμμής, η οποία είναι απαραίτητο να γίνεται σειριακά προκειμένου να έχουμε σωστό αποτέλεσμα και άρα πρέπει να βεβαιωθούμε ότι δεν εκτελούν τύπωμα εξόδου δύο thread με λανθασμένη σειρά ή σε επικαλυπτόμενες χρονικές στιγμές. Εφόσον λοιπόν το τμήμα υπολογισμού βρίσκεται εκτός του κρίσιμου τμήματος, όλα τα thread μπορούν να υπολογίζουν γραμμές ταυτόχρονα, κάτι που αυξάνει την ταχύτητα του προγράμματος.

4) Κατά την εκτέλεση του προγράμματος, όταν πατηθεί `Ctrl-C` δηλαδή σταλεί στη διεργασία ένα σήμα SIGINT με αποτέλεσμα να σταματήσει την εκτέλεση και να εμφανίζει στη συνέχεια το terminal οποιοδήποτε output με το χρώμα που έχει υπολογίσει το πρόγραμμά μας τελευταίο. Αυτό φαίνεται στο παρακάτω παράδειγμα όπου την ώρα που τυπωνόταν το @ πατήσαμε Ctrl-C και στην συνέχεια εμφανίζονται όλα με μπλε:



Προκειμένου να αποφύγουμε την κατάσταση αυτή, πρέπει να υλοποιήσουμε ενα signal handler – που θα εκτελεί κάθε φόρα που κάνει catch το εν λόγω σήμα – μια ρουτίνα που θα καλεί τη `reset_xterm_color ()` και μετά να τερματίζει τη λειτουργία του προγράμματος με κλήση της exit (), όπως ο παρακάτω:

```
void Isinterrupt(){
        reset_xterm_color(1);
        printf("There is Cntr-C \n");
        exit(1);
}
```

Η χρήση αυτού ως handler για το σήμα SIGINT ορίζεται με την εντολή signal(SIGINT,Isinterrupt);

Ο κώδικας του ερωτήματος αυτού για τις παράλληλες διεργασίες είναι:

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>
#include "mandel-lib.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 MANDEL_MAX_ITERATION 100000
/***************************
 * Compile-time parameters *
 ***************************/
struct thread_info_struct {
        pthread_t threadid; /* POSIX thread id, as returned by the library */
        int N; /* Application-defined thread id */

};
int NTHREADS;
/*
 * 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.
 */
sem_t *sem;
struct thread_info_struct *thrptr;
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(int fd, int line, int N)
{
        /*
         * A temporary array, used to hold color values for the line being drawn
         */
        int color_val[x_chars];
        compute_mandel_line(line, color_val);
        sem_wait(&sem[((line)%N)]);
        output_mandel_line(fd, color_val);
        sem_post(&sem[((line+1)%N)]);
}
int safe_fn(char *s)
{
        long l;
        char* endp;

        l = strtol(s, &endp, 10);
        if (s != endp && *endp == '\0') {
                return l;
        }
        else
                return -1;
}

void *thread_solution(void *arg)
{
        struct thread_info_struct *thr = arg;
        int line;
        for (line = thr->N; line < y_chars; line+= NTHREADS) {
                compute_and_output_mandel_line(1, line, NTHREADS);
        }
        return NULL;
}

//We can put here the function defined  in the asnwer of the 4th question
int main(int argc, char *argv[])
{
        int ret;
        if (argc != 2){
                perror("Heyyy, You must type the following structure ./mandel <N_THREADS>\n");
                exit(1);
        }

        NTHREADS = safe_fn(argv[1]);
        if (NTHREADS <= 0) {
                fprintf(stderr, "%s not valid for 'NTHREADS'\n", argv[2]);
                exit(2);
        }
```

```
        printf("Running for N_THREADS = %d\n", NTHREADS);
        printf("Lets create the threads\n");
      if ((thrptr = malloc(NTHREADS*sizeof(*thrptr))) == NULL) {
                              fprintf(stderr, "We have problem in allocation of %zd bytes \
n",NTHREADS*sizeof(*thrptr));
          exit(1);
      }
        printf("threads are OK\n");
        printf("Lets create semaphores\n");
        if ((sem = malloc(NTHREADS*sizeof(*sem))) == NULL) {
      fprintf(stderr, "We have problem in allocation of  %zd bytes \n",NTHREADS*sizeof(*sem));
        exit(1);
      }
        printf("sem is OK\n");
        int i;
        // Initializing semaphores
        for (i = 0; i < NTHREADS; i++){
                sem_init(&sem[i], 0, 0);
        }
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor '1', i.e., standard output.
         */
        // Incrementing the first semaphore
        sem_init(&sem[0], 0, 1);
        for (i = 0; i < NTHREADS; i++) {
                thrptr[i].N = i;

                /* Spawn new thread(s) */
                ret = pthread_create(&thrptr[i].threadid, NULL, thread_solution, &thrptr[i]);
                if (ret) {
                        perror_pthread(ret, "pthread_create");
                        exit(1);
                }
        }
        /*
         * Wait threads to terminate
         */
        for (i = 0; i < NTHREADS; i++) {
                ret = pthread_join(thrptr[i].threadid, NULL);
                if (ret) {
                        perror_pthread(ret, "pthread_join");
                        exit(1);
                }
        }

        reset_xterm_color(1);
        return 0;
}
```