ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ

ΣΧΟΛΉ ΗΛΕΚΤΡΟΛΌΓΩΝ ΜΗΧΑΝΙΚΏΝ ΚΑΙ ΜΗΧΑΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΉΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

(2019-2020)

3η ΕΡΓΑΣΤΗΡΙΑΚΉ ΑΝΑΦΟΡΆ

Ομάδα: oslaba36

Ονοματεπώνυμο: Χρήστος Τσούφης – 03117176

Νίκος Χαράλαμπος Χαιρόπουλος – 03119507

Εκτέλεση Εργαστηρίου: 04/05/2020

Σημείωση: Η άσκηση αυτή επιλύθηκε αρχικά ατομικά από τον καθένα μας για την καλύτερη κατανόηση των εννοιών. Παρακάτω παρουσιάζεται η αποδοτικότερη από τις δύο λύσεις.

Άσκηση 3: Συγχρονισμός

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 Ν φορές την τιμή της μεταβλητής val κατά 1, το δεύτερο τη μειώνει Ν 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;
  Αντιγράφοντας σε ένα νέο directory τα αρχεία simplesync.c , pthread-test.c και το κατάλληλα τροποποιημένο Makefile, και προσπαθώντας να εκτελεστεί η εντολή ./ simplesync.c και
  ./ pthread-test.c αντίστοιχα, βγαίνει το εξής μήνυμα: Permission denied
  Βρίσκονται στον φάκελο intro.
- 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!

Ο κώδικας για την pthread-test.c :

```c
/*
 * pthread-test.c
 *
 * A test program for POSIX Threads.
 *
 * 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>

#define VAL1 1.5
#define VAL2 4.0

/*
 * 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)

/*
 * A (distinct) instance of this structure
 * is passed to each thread
 */
struct thread_info_struct {
    pthread_t tid; /* POSIX thread id, as returned by the library */

    double *arr; /* Pointer to array to manipulate */
    int len;
    double mul;

    int thrid; /* Application-defined thread id */
    int thrcnt;
};

int safe_atoi(char *s, int *val)
```

```
{
    long l;
    char *endp;

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

void *safe_malloc(size_t size)
{
    void *p;

    if ((p = malloc(size)) == NULL) {
        fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
            size);
        exit(1);
    }

    return p;
}

void usage(char *argv0)
{
    fprintf(stderr, "Usage: %s thread_count array_size\n\n"
        "Exactly two argument required:\n"
        "    thread_count: The number of threads to create.\n"
        "    array_size: The size of the array to run with.\n",
        argv0);
    exit(1);
}


/* Start function for each thread */
void *thread_start_fn(void *arg)
{
    int i;

    /* We know arg points to an instance of thread_info_struct */
    struct thread_info_struct *thr = arg;

    fprintf(stderr, "Thread %d of %d. START.\n", thr->thrid, thr->thrcnt);

    for (i = thr->thrid; i < thr->len; i += thr->thrcnt)
        thr->arr[i] *= thr->mul;
```

```c
    fprintf(stderr, "Thread %d of %d. END.\n", thr->thrid, thr->thrcnt);

    return NULL;
}

int main(int argc, char *argv[])
{
    double *arr;
    int i, ret, arrsize, thrcnt;
    struct thread_info_struct *thr;

    /*
     * Parse the command line
     */
    if (argc != 3)
        usage(argv[0]);
    if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0) {
        fprintf(stderr, "`%s' is not valid for `thread_count'\n", argv[1]);
        exit(1);
    }
    if (safe_atoi(argv[2], &arrsize) < 0 || arrsize <= 0) {
        fprintf(stderr, "`%s' is not valid for `array_size'\n", argv[2]);
        exit(1);
    }

    /*
     * Allocate and initialize big array of doubles
     */
    arr = safe_malloc(arrsize * sizeof(*arr));
    for (i = 0; i < arrsize; i++)
        arr[i] = VAL1;

    /*
     * Multiply it by VAL2 using thrcnt threads, in parallel
     */
    thr = safe_malloc(thrcnt * sizeof(*thr));

    for (i = 0; i < thrcnt; i++) {
        /* Initialize per-thread structure */
        thr[i].arr = arr;
        thr[i].len = arrsize;
        thr[i].mul = VAL2;
        thr[i].thrid = i;
        thr[i].thrcnt = thrcnt;

        /* Spawn new thread */
        ret = pthread_create(&thr[i].tid, NULL, thread_start_fn, &thr[i]);
        if (ret) {
```

```
            perror_pthread(ret, "pthread_create");
            exit(1);
        }
    }

    /*
     * Wait for all threads to terminate
     */
    for (i = 0; i < thrcnt; i++) {
        ret = pthread_join(thr[i].tid, NULL);
        if (ret) {
            perror_pthread(ret, "pthread_join");
            exit(1);
        }
    }

    /*
     * Verify resulting values
     */
    for (i = 0; i < arrsize; i++)
        if (arr[i] != VAL1 * VAL2) {
            fprintf(stderr, "Internal error: arr[%d] = %f, not %f\n",
                i, arr[i], VAL1 * VAL2);
            exit(1);
    }

    printf("OK.\n");

    return 0;
}
```

Και ο πηγαίος κώδικας για την simplesync.c :

```
/*
 * simplesync.c
 *
 * A simple synchronization exercise.
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 *
 */

#include <linux/unistd.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```c
#include <pthread.h>
#include <sys/syscall.h>
#include <sched.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 mutexcount; // N


void *increase_fn(void *arg)
{
        // we input a mutex "door" in order to entry the room , if the mutex  is loc
ked the thread cannot entry the  increase function "door"

        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 */
                        __sync_add_and_fetch(ip,1);
                        /* ... */
                } else {
                        pthread_mutex_lock(&mutexcount);
                        /* ... */
                        /* You cannot modify the following line */
                        ++(*ip);
```

```
                        pthread_mutex_unlock(&mutexcount);

                        /* ... */
                }
        }

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

        // we input a mutex unlock , in order the thread free the mutex because it o
ut the door
        //pthread_exit(NULL); // it pass the door and exits (mutexes)
}



void *decrease_fn(void *arg)
{
        int i;
        volatile int *ip = arg;
        // we input a mutex "door" in order to entry the room , if the mutex  is loc
ked the thread cannot entry the  decrease  function "door"
        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 */
                    __sync_sub_and_fetch(ip,1);
                        /* ... */
                } else {
                        pthread_mutex_lock(&mutexcount);
                        /* ... */
                        /* You cannot modify the following line */
                        --(*ip);
                        pthread_mutex_unlock(&mutexcount);
                        /* ... */
                }
        }


        fprintf(stderr, "Done decreasing variable.\n");
        // we input a mutex unlock , in order the thread free the mutex because it o
ut the door
        // pthread_exit(NULL);
        return NULL;
}


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

```c
{

        pthread_mutex_init(&mutexcount,NULL); // tou lew oti einai ksekleidwto, to p
rwto lock tha petyxei dld
        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;
}
```

Και το Makefile:

```
CC = gcc
#compile multi-threaded applications with -pthread and when -g
#is use(for gdb info) use -O2 to optimize compilation again
CFLAGS = -Wall -O2 -pthread
LIBS =
all: pthread-test simplesync-atomic simplesync-mutex
##################### Pthread test ##############################
pthread-test: pthread-test.o
	$(CC) $(CFLAGS) -o pthread-test pthread-test.o
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
########################### Clean ##############################
clean:
	rm -f *.o pthread-test simplesync-atomic simplesync-mutex
```

<u>Ερωτήσεις</u>:

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;*

   Με την χρήση της εντολής time() μετρήθηκε ο χρόνος εκτέλεσης. Ο χρόνος εκτέλεσης των εκτελέσιμων που εκτελούν συγχρονισμό σε σχέση με τον χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό είναι συγκριτικά μεγαλύτερος. Αυτό αιτιολογείται από το γεγονός ότι ο συγχρονισμός (είτε atomic είτε mutex) περιέχει καθυστερήσεις (για την αντιμετώπιση rare conditions) οι οποίες δεν υφίστανται όταν τα threads δεν περιέχουν καμία δομή συγχρονισμού.

Εκτέλεση του αρχικού simplesync:

```
oslaba36@os-node1:~/exe3/exer3_1$ time ./simplesync
-bash: ./simplesync: No such file or directory

real    0m0.002s
user    0m0.000s
sys     0m0.000s
oslaba36@os-node1:~/exe3/exer3_1$
```

Εκτέλεση του simplesync-atomic:

```
oslaba36@os-node1:~/ask3/sync$ time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.

real    0m0.412s
user    0m0.808s
sys     0m0.004s
oslaba36@os-node1:~/ask3/sync$
```

Εκτέλεση του simplesync-mutex:

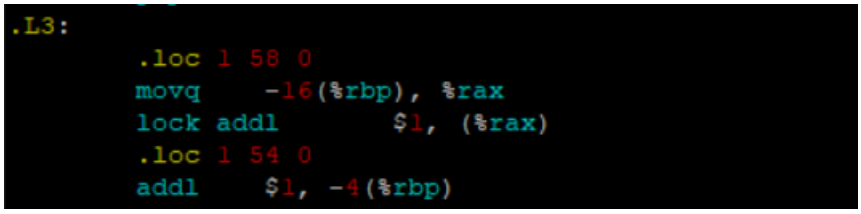
```
oslaba36@os-node1:~/ask3/sync$ time ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.

real    0m4.254s
user    0m4.660s
sys     0m3.164s
```

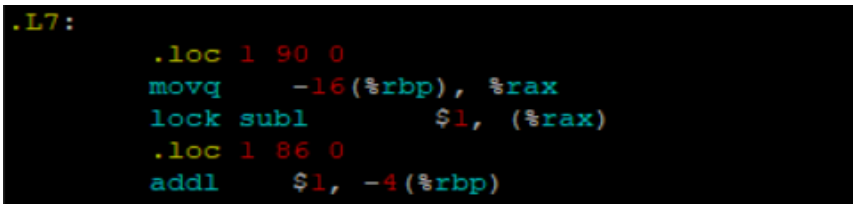
2. *Ποια μέθοδος συγχρονισμού είναι γρηγορότερη, η χρήση ατομικών λειτουργιών ή η χρήση POSIX mutexes; Γιατί;*

Παρατηρώντας τις παραπάνω εκτελέσεις και τα αποτελέσματά τους προκύπτει ότι η χρήση ατομικών λειτουργιών είναι γρηγορότερη. Αυτό αιτιολογείται από το γεγονός ότι οι ατομικές λειτουργίες υλοποιούνται κατευθείαν στο hardware, ενώ τα mutexes που περιλαμβάνουν την κλήση συναρτήσεων για την υλοποίησής τους. Εν κατακλείδι, τα atomic operations είναι πιο γρήγορα αν η “διαμάχη” μεταξύ των threads είναι ικανοποιητικά χαμηλά.

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.*

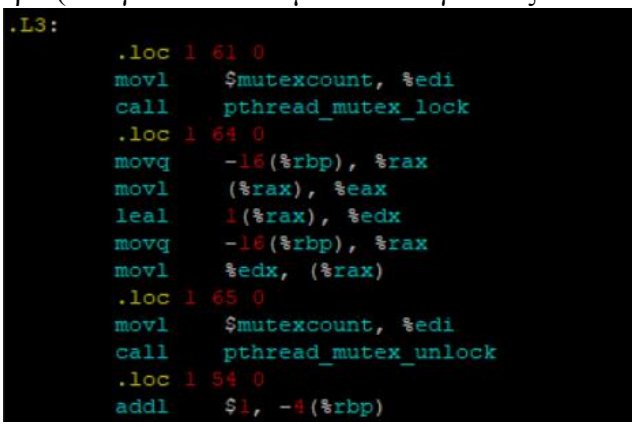
   Με την εκτέλεση της εντολής προκύπτει το εξής:
   ```
   gcc -DSYNC_ATOMIC=1 -S -g simplesync.c
   vi simplesync.s
   ```
   Μέσα στο assembly αρχείο εντοπίζονται οι σχετικές εντολές στις οποίες μεταφράζονται τα atomic operations. Αρχικά, η εντολή lock δίνει οδηγία στον επεξεργαστή να εκτελέσει την εντολή που ακολουθεί ως atomic operation . Οι σχετικές εντολές προς τον επεξεργαστή είναι lock addl και lock subl. Το .loc 1 58 0 και το .loc 1 90 0 είναι για την αντιστοιχία με κώδικα σε 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.*

   Με την εκτέλεση της εντολής προκύπτει το εξής:
   ```
   gcc -DSYNC_MUTEX=1 -S -g simplesync.c
   vi simplesync.s
   ```
   Μέσα στο assembly αρχείο εντοπίζονται οι σχετικές εντολές στις οποίες μεταφράζονται τα mutexes. Παρατηρώντας το αρχείο φαίνονται και οι λόγοι για τους οποίους η χρήση των mutexes καθυστερεί περισσότερο αφού για τα lock και unlock των mutexes της κάθε πράξης απαιτούνται δύο system calls κάθε φορά (ένα για το κλείδωμα και ένα για το ξεκλείδωμα).



```
.L7:
        .loc 1 93 0
        movl    $mutexcount, %edi
        call    pthread_mutex_lock
        .loc 1 96 0
        movq    -16(%rbp), %rax
        movl    (%rax), %eax
        leal    -1(%rax), %edx
        movq    -16(%rbp), %rax
        movl    %edx, (%rax)
        .loc 1 97 0
        movl    $mutexcount, %edi
        call    pthread_mutex_unlock
        .loc 1 86 0
        addl    $1, -4(%rbp)
```

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 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.

Ο κώδικας για το mandel.c:

```
/*
 * 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 <pthread.h>
#include <semaphore.h>
#include "mandel-lib.h"
#include <errno.h>
#include <time.h>
#include <sched.h>

#define MANDEL_MAX_ITERATION 100000
#define _GNU_SOURCE
/***************************
 * Compile-time parameters *
 ***************************/

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;

//we declare a pointer to semaphore
//sem_t keysem;
sem_t * sem_array;
int bound=0; //global , is protected via semaphore

 // we declare a struct in order to pass multiple arguments to the function from the
 thread
     struct arguments{
               int fd;
               int line;
               int N_SIZE;// the number of threads that are given from the user
               int thread_num;
    };

//END
```

```
/* 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;
        }
}

void output_mandel_line(int fd, int color_val[]){
/*
```

```
 * This function outputs an array of x_char color values
 * to a 256-color xterm.
 */
        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 * datatopass)
{

    struct arguments * dedomena = (struct arguments*) datatopass;

        int j    = dedomena->line;
        int fd   = dedomena->fd;
        int line = dedomena->line;
        int N    = dedomena->N_SIZE;
        int thread_code = dedomena->thread_num;
        int color_val[x_chars];

     while(j<y_chars){


        compute_mandel_line(line, color_val);           //to kanoun ola ta threads
, den exei lock
                                                          // printf("thread = %d  \
n",thread_code);

        // set the semahore
        sem_wait(&sem_array[j]);
        output_mandel_line(fd,color_val);
        line += N;
```

```
        bound++;

        if ( bound  >= y_chars ){
                break;
        }

        sem_post(&sem_array[j+1]);
        j=j+N;
    }

    //pthread_exit(NULL);
    return NULL;

}

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


char *  THREADS_NUM =(argv[1]); // the number of threads!
int N = atoi(THREADS_NUM);

/*we declare dynamicaly an array of semaphores , all of them are initialized to zero
 every thread coresponds to a semaphore , the n thread wakes up the n+1 semaphore
*/

int i=0;
 sem_array =(sem_t*) malloc(sizeof(sem_t)*50);
for(i=0 ; i < 50 ; i++ ) {
        sem_init(&sem_array[i],0,0);// ?
}
// set the first semaphore to 1 to open the "door" for the first thread

sem_post(&sem_array[0]);
//END

// we declare an array of threads dynamicaly:
        pthread_t  *id_array;
        id_array = (pthread_t*)malloc(sizeof(pthread_t)*N);
// END

//we declare an array of structs dynamicaly:
        struct arguments * data = (struct arguments *) malloc(sizeof(struct argument
s)*N);
// END
```

```
        //int lia;
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        for( i=0 ; i < N ; i++){
                (data+i)->N_SIZE = N;
                (data+i)->fd = 1;
                (data+i)->line=i;
                (data+i)->thread_num=i;

                int val1 = (data+i)->N_SIZE;
                int val2 = (data+i)->fd;
                int val3 = (data+i)->line;

                //printf( " INITIALIZATIONi :  N = %d , fd = %d , line = %d \n",val1
,val2,val3);
                pthread_create(id_array+i,NULL,compute_and_output_mandel_line,&data[
i]);
        }

//       for( i=0 ; i < N ; i++){
//           printf("THREAD CREATION \n");
//           pthread_create(id_array+i,NULL,compute_and_output_mandel_line,data+i);
//       }


        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor '1', i.e., standard output.
         */
//      for (line = 0; line < y_chars; line++) {
//              compute_and_output_mandel_line(1, line);
//      }
        //pthread_yield();// dinei mia protetraiotha se aytous poy perimenoun
        for( i = 0 ; i < N ; i++){
        pthread_join(*(id_array+i), NULL);
        }
        reset_xterm_color(1);
        return 0;
}
```

Ο κώδικας για το mandel-lib.c:

```c
/*
 * mandel-lib.c
 *
 * A library with useful functions
 * for computing the Mandelbrot Set and handling a 256-color xterm.
 *
 */

#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#include "mandel-lib.h"

/*****************************************
 *                                       *
 * Functions to manage a 256-color xterm *
 *                                       *
 *****************************************/

/* 3 functions to convert between RGB colors and the corresponding xterm-256 values
 * Wolfgang Frisch, xororand@frexx.de */


// whole colortable, filled by maketable()
static int initialized=0;
static unsigned char colortable[254][3];

// the 6 value iterations en the xterm color cube
static const unsigned char valuerange[] = { 0x00, 0x5F, 0x87, 0xAF, 0xD7, 0xFF };

// 16 basic colors
static const unsigned char basic16[16][3] =
{
    { 0x00, 0x00, 0x00 }, // 0
    { 0xCD, 0x00, 0x00 }, // 1
    { 0x00, 0xCD, 0x00 }, // 2
    { 0xCD, 0xCD, 0x00 }, // 3
    { 0x00, 0x00, 0xEE }, // 4
    { 0xCD, 0x00, 0xCD }, // 5
    { 0x00, 0xCD, 0xCD }, // 6
    { 0xE5, 0xE5, 0xE5 }, // 7
    { 0x7F, 0x7F, 0x7F }, // 8
    { 0xFF, 0x00, 0x00 }, // 9
```

```
    { 0x00, 0xFF, 0x00 }, // 10
    { 0xFF, 0xFF, 0x00 }, // 11
    { 0x5C, 0x5C, 0xFF }, // 12
    { 0xFF, 0x00, 0xFF }, // 13
    { 0x00, 0xFF, 0xFF }, // 14
    { 0xFF, 0xFF, 0xFF }  // 15
};

// convert an xterm color value (0-253) to 3 unsigned chars rgb
static void xterm2rgb(unsigned char color, unsigned char* rgb)
{
    // 16 basic colors
    if(color<16)
    {
        rgb[0] = basic16[color][0];
        rgb[1] = basic16[color][1];
        rgb[2] = basic16[color][2];
    }

    // color cube color
    if(color>=16 && color<=232)
    {
        color-=16;
        rgb[0] = valuerange[(color/36)%6];
        rgb[1] = valuerange[(color/6)%6];
        rgb[2] = valuerange[color%6];
    }

    // gray tone
    if(color>=233 && color<=253)
    {
        rgb[0]=rgb[1]=rgb[2] = 8+(color-232)*0x0a;
    }
}

// fill the colortable for use with rgb2xterm
static void maketable()
{
    unsigned char c, rgb[3] = {0, 0, 0};
    for(c=0;c<=253;c++)
    {
        xterm2rgb(c,rgb);
        colortable[c][0] = rgb[0];
        colortable[c][1] = rgb[1];
        colortable[c][2] = rgb[2];
    }
}
```

```
// selects the nearest xterm color for a 3xBYTE rgb value
static unsigned char rgb2xterm(unsigned char* rgb)
{
    unsigned char c, best_match=0;
    double d, smallest_distance;

    if(!initialized)
        maketable();

    smallest_distance = 10000000000.0;

    for(c=0;c<=253;c++)
    {
        d = pow(colortable[c][0]-rgb[0],2.0) +
            pow(colortable[c][1]-rgb[1],2.0) +
            pow(colortable[c][2]-rgb[2],2.0);
        if(d<smallest_distance)
        {
            smallest_distance = d;
            best_match=c;
        }
    }

    return best_match;
}


/****************************************
 *                                      *
 * A nice 256-color palette for drawing *
 * the Mandelbrot Set.                  *
 *                                      *
 ****************************************/

static struct { double red; double green; double blue; } mandel256[] = {
    {0.000,0.000,0.734},
    {0.000,0.300,0.734},
    {0.000,0.734,0.000},
    {0.734,0.734,0.000},
    {0.734,0.000,0.000},
    {0.734,0.000,0.734},
    {0.000,0.734,0.734},
    {0.750,0.750,0.750},
    {0.750,0.859,0.750},
    {0.641,0.781,0.938},
    {0.500,0.000,0.000},
    {0.000,0.500,0.000},
    {0.500,0.500,0.000},
    {0.000,0.000,0.500},
```

```
    {0.500,0.000,0.500},
    {0.000,0.500,0.500},
    {0.234,0.359,0.234},
    {0.359,0.359,0.234},
    {0.484,0.359,0.234},
    {0.609,0.359,0.234},
    {0.734,0.359,0.234},
    {0.859,0.359,0.234},
    {0.984,0.359,0.234},
    {0.234,0.484,0.234},
    {0.359,0.484,0.234},
    {0.484,0.484,0.234},
    {0.609,0.484,0.234},
    {0.734,0.484,0.234},
    {0.859,0.484,0.234},
    {0.984,0.484,0.234},
    {0.234,0.609,0.234},
    {0.359,0.609,0.234},
    {0.484,0.609,0.234},
    {0.609,0.609,0.234},
    {0.734,0.609,0.234},
    {0.859,0.609,0.234},
    {0.984,0.609,0.234},
    {0.234,0.734,0.234},
    {0.359,0.734,0.234},
    {0.484,0.734,0.234},
    {0.609,0.734,0.234},
    {0.734,0.734,0.234},
    {0.859,0.734,0.234},
    {0.984,0.734,0.234},
    {0.234,0.859,0.234},
    {0.359,0.859,0.234},
    {0.484,0.859,0.234},
    {0.609,0.859,0.234},
    {0.734,0.859,0.234},
    {0.859,0.859,0.234},
    {0.984,0.859,0.234},
    {0.234,0.984,0.234},
    {0.359,0.984,0.234},
    {0.484,0.984,0.234},
    {0.609,0.984,0.234},
    {0.734,0.984,0.234},
    {0.859,0.984,0.234},
    {0.984,0.984,0.234},
    {0.234,0.234,0.359},
    {0.359,0.234,0.359},
    {0.484,0.234,0.359},
    {0.609,0.234,0.359},
```

```
    {0.734,0.234,0.359},
    {0.859,0.234,0.359},
    {0.984,0.234,0.359},
    {0.234,0.359,0.359},
    {0.359,0.359,0.359},
    {0.484,0.359,0.359},
    {0.609,0.359,0.359},
    {0.734,0.359,0.359},
    {0.859,0.359,0.359},
    {0.984,0.359,0.359},
    {0.234,0.484,0.359},
    {0.359,0.484,0.359},
    {0.484,0.484,0.359},
    {0.609,0.484,0.359},
    {0.734,0.484,0.359},
    {0.859,0.484,0.359},
    {0.984,0.484,0.359},
    {0.234,0.609,0.359},
    {0.359,0.609,0.359},
    {0.484,0.609,0.359},
    {0.609,0.609,0.359},
    {0.734,0.609,0.359},
    {0.859,0.609,0.359},
    {0.984,0.609,0.359},
    {0.234,0.734,0.359},
    {0.359,0.734,0.359},
    {0.484,0.734,0.359},
    {0.609,0.734,0.359},
    {0.734,0.734,0.359},
    {0.859,0.734,0.359},
    {0.984,0.734,0.359},
    {0.234,0.859,0.359},
    {0.359,0.859,0.359},
    {0.484,0.859,0.359},
    {0.609,0.859,0.359},
    {0.734,0.859,0.359},
    {0.859,0.859,0.359},
    {0.984,0.859,0.359},
    {0.234,0.984,0.359},
    {0.359,0.984,0.359},
    {0.484,0.984,0.359},
    {0.609,0.984,0.359},
    {0.734,0.984,0.359},
    {0.859,0.984,0.359},
    {0.984,0.984,0.359},
    {0.234,0.234,0.484},
    {0.359,0.234,0.484},
    {0.484,0.234,0.484},
```

```
    {0.609,0.234,0.484},
    {0.734,0.234,0.484},
    {0.859,0.234,0.484},
    {0.984,0.234,0.484},
    {0.234,0.359,0.484},
    {0.359,0.359,0.484},
    {0.484,0.359,0.484},
    {0.609,0.359,0.484},
    {0.734,0.359,0.484},
    {0.859,0.359,0.484},
    {0.984,0.359,0.484},
    {0.234,0.484,0.484},
    {0.359,0.484,0.484},
    {0.484,0.484,0.484},
    {0.609,0.484,0.484},
    {0.734,0.484,0.484},
    {0.859,0.484,0.484},
    {0.984,0.484,0.484},
    {0.234,0.609,0.484},
    {0.359,0.609,0.484},
    {0.484,0.609,0.484},
    {0.609,0.609,0.484},
    {0.734,0.609,0.484},
    {0.859,0.609,0.484},
    {0.984,0.609,0.484},
    {0.234,0.734,0.484},
    {0.359,0.734,0.484},
    {0.484,0.734,0.484},
    {0.609,0.734,0.484},
    {0.734,0.734,0.484},
    {0.859,0.734,0.484},
    {0.984,0.734,0.484},
    {0.234,0.859,0.484},
    {0.359,0.859,0.484},
    {0.484,0.859,0.484},
    {0.609,0.859,0.484},
    {0.734,0.859,0.484},
    {0.859,0.859,0.484},
    {0.984,0.859,0.484},
    {0.234,0.984,0.484},
    {0.359,0.984,0.484},
    {0.484,0.984,0.484},
    {0.609,0.984,0.484},
    {0.734,0.984,0.484},
    {0.859,0.984,0.484},
    {0.984,0.984,0.484},
    {0.234,0.234,0.609},
    {0.359,0.234,0.609},
```

```
    {0.484,0.234,0.609},
    {0.609,0.234,0.609},
    {0.734,0.234,0.609},
    {0.859,0.234,0.609},
    {0.984,0.234,0.609},
    {0.234,0.359,0.609},
    {0.359,0.359,0.609},
    {0.484,0.359,0.609},
    {0.609,0.359,0.609},
    {0.734,0.359,0.609},
    {0.859,0.359,0.609},
    {0.984,0.359,0.609},
    {0.234,0.484,0.609},
    {0.359,0.484,0.609},
    {0.484,0.484,0.609},
    {0.609,0.484,0.609},
    {0.734,0.484,0.609},
    {0.859,0.484,0.609},
    {0.984,0.484,0.609},
    {0.234,0.609,0.609},
    {0.359,0.609,0.609},
    {0.484,0.609,0.609},
    {0.609,0.609,0.609},
    {0.734,0.609,0.609},
    {0.859,0.609,0.609},
    {0.984,0.609,0.609},
    {0.234,0.734,0.609},
    {0.359,0.734,0.609},
    {0.484,0.734,0.609},
    {0.609,0.734,0.609},
    {0.734,0.734,0.609},
    {0.859,0.734,0.609},
    {0.984,0.734,0.609},
    {0.234,0.859,0.609},
    {0.359,0.859,0.609},
    {0.484,0.859,0.609},
    {0.609,0.859,0.609},
    {0.734,0.859,0.609},
    {0.859,0.859,0.609},
    {0.984,0.859,0.609},
    {0.234,0.984,0.609},
    {0.359,0.984,0.609},
    {0.484,0.984,0.609},
    {0.609,0.984,0.609},
    {0.734,0.984,0.609},
    {0.859,0.984,0.609},
    {0.984,0.984,0.609},
    {0.234,0.234,0.734},
```

```
    {0.359,0.234,0.734},
    {0.484,0.234,0.734},
    {0.609,0.234,0.734},
    {0.734,0.234,0.734},
    {0.859,0.234,0.734},
    {0.984,0.234,0.734},
    {0.234,0.359,0.734},
    {0.359,0.359,0.734},
    {0.484,0.359,0.734},
    {0.609,0.359,0.734},
    {0.734,0.359,0.734},
    {0.859,0.359,0.734},
    {0.984,0.359,0.734},
    {0.234,0.484,0.734},
    {0.359,0.484,0.734},
    {0.484,0.484,0.734},
    {0.609,0.484,0.734},
    {0.734,0.484,0.734},
    {0.859,0.484,0.734},
    {0.984,0.484,0.734},
    {0.234,0.609,0.734},
    {0.359,0.609,0.734},
    {0.484,0.609,0.734},
    {0.609,0.609,0.734},
    {0.734,0.609,0.734},
    {0.859,0.609,0.734},
    {0.984,0.609,0.734},
    {0.234,0.734,0.734},
    {0.359,0.734,0.734},
    {0.484,0.734,0.734},
    {0.609,0.734,0.734},
    {0.734,0.734,0.734},
    {0.859,0.734,0.734},
    {0.984,0.734,0.734},
    {0.234,0.859,0.734},
    {0.359,0.859,0.734},
    {0.484,0.859,0.734},
    {0.609,0.859,0.734},
    {0.734,0.859,0.734},
    {0.859,0.859,0.734},
    {0.984,0.969,0.938},
    {0.625,0.625,0.641},
    {0.500,0.500,0.500},
    {0.984,0.000,0.000},
    {0.000,0.984,0.000},
    {0.984,0.984,0.000},
    {0.000,0.000,0.984},
    {0.984,0.000,0.984},
```

```
    {0.000,0.984,0.984},
    {0.000,0.000,0.000}
};

/****************************************
 *                                      *
 * Functions to compute the Mandelbrot set *
 *                                      *
 ****************************************/

/*
 * This function takes a (x,y) point on the complex plane
 * and uses the escape time algorithm to return a color value
 * used to draw the Mandelbrot Set.
 */
int mandel_iterations_at_point(double x, double y, int max)
{
    double x0 = x;
    double y0 = y;
    int iter = 0;

    while ( (x * x + y * y <= 4) && iter < max) {
        double xt = x * x - y * y + x0;
        double yt = 2 * x * y + y0;

        x = xt;
        y = yt;

        ++iter;
    }

    return iter;
}

/*
 * This function takes a color value as returned
 * by mandelbrot_iterations() and uses the 256-color
 * palette defined above to return an approximation for 256-color
 * xterms.
 */
unsigned char xterm_color(int color_val)
{
    unsigned char rgb[3];

    if (color_val > 255)
        color_val = 255;

    rgb[0] = 255.0 * mandel256[color_val].red;
```

```
    rgb[1] = 255.0 * mandel256[color_val].green;
    rgb[2] = 255.0 * mandel256[color_val].blue;
    color_val = rgb2xterm(rgb);

    assert(0 <= color_val && color_val <= 255);
    return color_val;
}

/*
 * Insist until all count bytes beginning at
 * address buff have been written to file descriptor fd.
 */
ssize_t insist_write(int fd, const char *buf, size_t count)
{
    ssize_t ret;
    size_t orig_count = count;

    while (count > 0) {
        ret = write(fd, buf, count);
        if (ret < 0)
            return ret;
        buf += ret;
        count -= ret;
    }

    return orig_count;
}

/*
 * This function outputs the proper control sequence
 * to change the current color of a 256-color xterm.
 */
void set_xterm_color(int fd, unsigned char color)
{
    char buf[100];
    snprintf(buf, 100, "\033[38;5;%dm", color);

    if (insist_write(fd, buf, strlen(buf)) != strlen(buf)) {
        perror("set_xterm_color: insist_write");
        exit(1);
    }
}

/*
 * Reset all character attributes before leaving,
 * to ensure the prompt is not drawn in a funny color
 */
void reset_xterm_color(int fd)
```

```
{
    if (insist_write(fd, "\033[0m", 4) != 4) {
        perror("reset_xterm_color: insist_write");
        exit(1);
    }
}
```

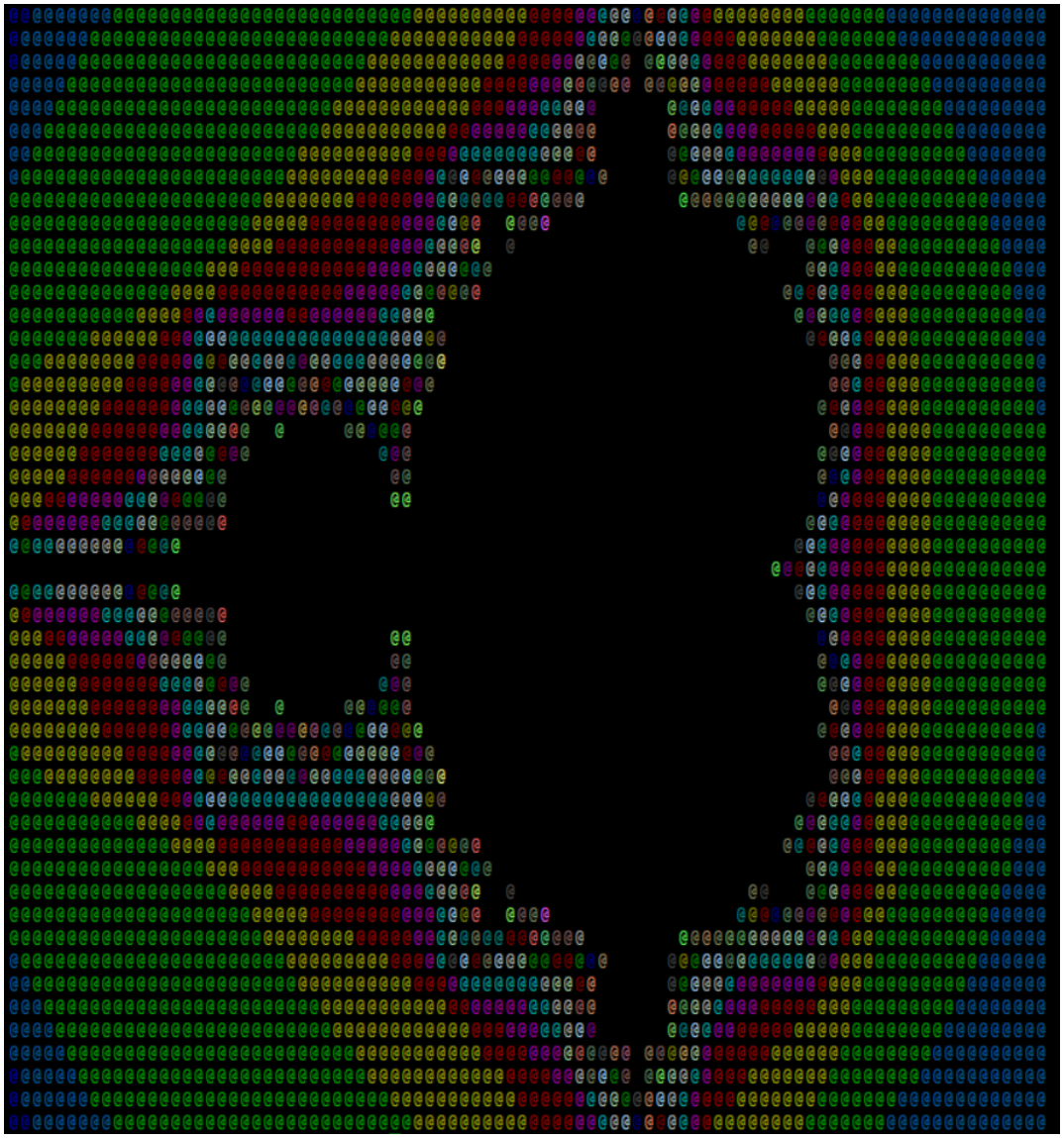
Και το mandel-lib.h :

```
/*
 * mandel-lib.h
 *
 * A library with useful functions
 * for computing the Mandelbrot Set and handling a 256-color xterm.
 *
 */

#ifndef MANDEL_LIB_H__
#define MANDEL_LIB_H__

/* Function prototypes */
int mandel_iterations_at_point(double x, double y, int max);
unsigned char xterm_color(int color_val);
ssize_t insist_write(int fd, const char *buf, size_t count);
void set_xterm_color(int fd, unsigned char color);
void reset_xterm_color(int fd);

#endif /* MANDEL_LIB_H__ */
```

Και το Makefile :

```
CC = gcc

#compile multi-threaded applications with -pthread and when -g

#is use(for gdb info) use -O2 to optimize compilation again

CFLAGS = -Wall -O2 -pthread

all: mandel

##################### Mandelbrot ##############################

mandel: mandel.o mandel-lib.o

	$(CC) $(CFLAGS) -o mandel mandel.o mandel-lib.o

mandel.o: mandel.c

	$(CC) $(CFLAGS) -c -o mandel.o mandel.c

mandel-lib.o: mandel-lib.c

	$(CC) $(CFLAGS) -c -o mandel-lib.o mandel-lib.c

########################## Clean ################################

clean:

	rm -f *.o mandel
```

Και για το output εκτελώντας την εντολή ./mandel και συμπληρώνοντας δίπλα τον αριθμό των threads (π.χ. ./mandel 2) προκύπτει:



<u>Ερωτήσεις</u>:

1. *Πόσοι σημαφόροι χρειάζονται για το σχήμα συγχρονισμού που υλοποιείτε;*

   Κρίνεται απαραίτητη η χρήση ίδιου αριθμού σημαφόρων και threads.
   Με άλλα λόγια, όσα lines είναι, τόσοι σημαφόροι χρειάζονται.

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.*

Εκτελώντας την εντολή time ./mandel και συμπληρώνοντας δίπλα τον αριθμό των threads (π.χ. ./mandel 2) προκύπτει:
Ενδεικτικά για N_THREADS = 1, δηλαδή σειριακή εκτέλεση :

```
real    0m1.019s
user    0m0.996s
sys     0m0.004s
oslaba36@os-node1:~/ask3/sync$
```

Ενδεικτικά για N_THREADS = 2:

```
real    0m0.519s
user    0m0.976s
sys     0m0.024s
oslaba36@os-node1:~/ask3/sync$
```

Ενδεικτικά για N_THREADS = 3:

```
real    0m0.351s
user    0m0.964s
sys     0m0.036s
oslaba36@os-node1:~/ask3/sync$
```

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;*

   Το πρόγραμμα εμφανίζει επιτάχυνση, διότι το κάθε thread υπολογίζει τις γραμμές που του αναλογούν σε διαφορετικό πυρήνα, με αποτέλεσμα οι υπολογισμοί να γίνονται παράλληλα και συνεπώς γρηγορότερα. Το κρίσιμο τμήμα είναι μόνο ο κώδικας που τυπώνει το αποτέλεσμα. Εάν στο κρίσιμο τμήμα είχαν συμπεριληφθεί οι υπολογισμοί, η χρήση των threads θα επιτάχυνε πολύ λιγότερο την διαδικασία, αφού αυτή θα πλησίαζε περισσότερο τη σειριακή εκτέλεση.
   Εάν το compute τοποθετηθεί στο κρίσιμο τμήμα (το οποίο οργανώνει τους σημαφόρους και εκτυπώνει) θα γίνει σειριακή εκτέλεση.

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;*

   Όπως φαίνεται και παρακάτω, πατώντας Ctrl + C κατά της διάρκεια της εκτέλεσης, το τερματικό παραμένει στο προηγούμενο χρώμα στο οποίο είχε γίνει set. Με κατάλληλη τροποποίηση του κώδικα, όταν θα πιάνει το σήμα SIGINT, θα κάνει reset το χρώμα του terminal.

```
// Out of main
void int_handler(int signum) {
     printf("Caught SIGINT signal! Reseting terminal color and exiting!\n");
     reset_xterm_color(1);
     exit(1);
}

// In main
   struct sigaction action;
   action.sa_handler = int_handler;
   sigaction (SIGINT, &action, NULL);
```