

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ
ΑΝΑΦΟΡΑ

Άσκηση 3

Αρετή Μέη 03120062
Ίρις-Ελευθερία Παλιατσού 03120639

1.1 Συγχρονισμός σε υπάρχοντα κώδικα

- Το Makefile που χρησιμοποιήθηκε για την μεταγλώττιση και εκτέλεση του προγράμματος είναι το εξής:

```
#  
# Makefile  
  
CC = gcc  
  
# CAUTION: Always use '-pthread' when compiling POSIX threads-based  
# applications, instead of linking with "-lpthread" directly.  
CFLAGS = -Wall -O2 -pthread  
LIBS =  
  
all: pthread-test sync-mutex sync-atomic simplesync-mutex simplesync-atomic kgar  
ten mandel mandel-sem mandel-cond  
  
## 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.s: simplesync.c  
	$(CC) $(CFLAGS) -DSYNC_MUTEX -S -g -o simplesync-mutex.s simplesync.c  
  
simplesync-atomic.s: simplesync.c  
	$(CC) $(CFLAGS) -DSYNC_ATOMIC -S -g -o simplesync-atomic.s simplesync.c  
  
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  
  
  
sync-mutex: sync-mutex.o  
	$(CC) $(CFLAGS) -o sync-mutex sync-mutex.o $(LIBS)
```

```

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

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

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

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

kgarten.o: kgarten.c
          $(CC) $(CFLAGS) -c -o kgarten.o kgarten.c

## Mandel
mandel: mandel-lib.o mandel.o
        $(CC) $(CFLAGS) -o mandel mandel-lib.o mandel.o $(LIBS)

mandel-sem: mandel-lib.o mandel-sem.o
           $(CC) $(CFLAGS) -o mandel-sem mandel-lib.o mandel-sem.o $(LIBS)

mandel-cond: mandel-lib.o mandel-cond.o
            $(CC) $(CFLAGS) -o mandel-cond mandel-lib.o mandel-cond.o $(LIBS)

mandel-lib.o: mandel-lib.h mandel-lib.c
              $(CC) $(CFLAGS) -c -o mandel-lib.o mandel-lib.c $(LIBS)

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

mandel-sem.o: mandel-sem.c
              $(CC) $(CFLAGS) -c -o mandel-sem.o mandel-sem.c $(LIBS)

mandel-cond.o: mandel-cond.c
                $(CC) $(CFLAGS) -c -o mandel-cond.o mandel-cond.c $(LIBS)

clean:
      rm -f *.s *.o pthread-test simplesync-{atomic,mutex} kgarten mandel

```

Παρατηρούμε ότι δημιουργούνται δύο εκτελέσιμα, sync-mutex και sync-atomic, ανάλογα με τα flags -DSYNC-MUTEX & -DSYNC-ATOMIC αντίστοιχα. Ο compiler κάνει predifine τις macro SYNC_ATOMIC και SYNC_MUTEX και ανάλογα με το ποια έχει γίνει predifine θέτει την USE_ATOMIC_OPS 0 ή 1, και στην συνέχεια αυτό χρησιμοποιείται στο if statement, προκειμένου να καθοριστεί ποιο από τα κομμάτι κώδικα if ή else θα εκτελεστεί.

- Ο νέος κώδικας είναι ο εξής:

```
/*  
 * 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  
  
static 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++) {  
        1,2
```

```
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); //++(*ip);
        /* ... */
    } 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 */
            sync_sub_and_fetch(ip, 1); //--(*ip);
            /* ... */
        } 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;
}

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

```

```

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;
}

/*
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
*/

```

Για να συγχρονίζεται η εκτέλεση των δύο νημάτων με τη χρήση POSIX mutexes, έχουμε στον παραπάνω κώδικα:

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

και

```
    } else {
        /* ... */
        pthread_mutex_lock(&mutex);
        /* You cannot modify the following line */
        --(*ip);
        pthread_mutex_unlock(&mutex);
        /* ... */
    }
```

Η ορθή λειτουργία του προγράμματος επιβεβαιώνεται με την εκτύπωση της τελικής τιμής val, μετάτον συγχρονισμό των 2 νημάτων. Παρατηρούμε πως ενώ πριν τον συγχρονισμό η τιμή της val ήταν διάφορη του 0(γεγονός που οφείλεται στην ασυγχρόνιστη λειτουργία των νημάτων και τα διάφορα race conditions που προκύπτουν), αφού επιτευχθεί ο συγχρονισμός, η τιμή val=0.

- Το κομμάτι του νέου κώδικα ώστε η εκτέλεση των δύο νημάτων να συγχρονίζεται με χρήση ατομικών λειτουργιών του GCC είναι:

```
if (USE_ATOMIC_OPS) {
    /* ... */
    /* You can modify the following line */
    sync_add_and_fetch(ip, 1); //++(*ip);
    /* ... */
}
```

και

```
(i = 0; i < N; i++) {
    if (USE_ATOMIC_OPS) {
        /* ... */
        /* You can modify the following line */
        sync_sub_and_fetch(ip, 1); //--(*ip);
        /* ... */
    }
}
```

Η ορθή λειτουργία επιβεβαιώνεται και πάλι με τη σωστή τιμή της val, την οποία και πάλι θέλουμε να είναι 0.

Ερωτήσεις:

1. Χρησιμοποιήθηκε η εντολή time προκειμένου να μετρηθεί ο χρόνος εκτέλεσης των εκτελέσιμων με και χωρίς συγχρονισμό. Παρατηρούμε ότι ο χρόνος του αρχικού εκτελέσιμου χωρις τον συγχρονισμό είναι μικρότερος από τον χρόνο εκτέλεσης των simplesync-atomic και simplesync-mutex. Αυτό συμβαίνει διότι στο αρχικό μας εκτελέσιμο τα νήματα εκτελούνται παράλληλα επομένως ο συνολικός χρόνος είναι μικρότερος, ωστόσο δεν υπάρχει η σωστή σειρά εκτέλεσης τους, άρα και το αποτέλεσμα που προκύπτει είναι λάθος. Αντίθετα στα εκτελέσιμα με συγχρονισμό, ο χρόνος είναι μεγαλύτερος αφού το critical section εκτελείται σειριακά από τα νήματα, για να επιτευχθεί ο συγχρονισμός.

```

gcc -Wall -O2 penitent -o sync-atomic sync-atomic.c
oslab25@orion:~/exercise3$ time ./sync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -2812861.

real    0m0.301s
user    0m0.136s
sys     0m0.004s
oslab25@orion:~/exercise3$ 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    0m1.307s
user    0m0.652s
sys     0m0.012s
oslab25@orion:~/exercise3$ 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    0m1.979s
user    0m0.912s
sys     0m0.024s

```

2. Όπως παρατηρούμε παραπάνω, γρηγορότερη είναι η χρήση των gcc atomic operations. Αυτό συμβαίνει γιατί σε αυτήν την περίπτωση η διαχείριση των νημάτων γίνεται απευθείας από το υλικό, και έτσι οι εντολές του critical section που έχουμε ορίσει για τα νήματα εκτελούνται ατομικά.
3. Προσθέτουμε στο Makefile:

```

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

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

```

Για την Tincrease:

```

.L2:
    .loc 1 51 0
    lock addl    $1, (%rbx)
.LVL5:

```

Για την Tdecrease:

```
.L7:
```

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

4. Όσον αφορά την χρήση των POSEX mutexes, έχουμε τις εντολές:

Tincrease:

```
.L2:
```

```
    .loc 1 55 0
    movl    $mutex, %edi
    call    pthread_mutex_lock
```

```
.LVL4:
```

```
    .loc 1 57 0
    movl    0(%rbp), %eax
    .loc 1 58 0
    movl    $mutex, %edi
    .loc 1 57 0
    addl    $1, %eax
    movl    %eax, 0(%rbp)
    .loc 1 58 0
    call    pthread_mutex_unlock
```

- 5.

Tdecrease:

```
.L7:
    .loc 1 81 0
    movl    $mutex, %edi
    call    pthread_mutex_lock
.LVL14:
    .loc 1 83 0
    movl    0(%rbp), %eax
    .loc 1 84 0
    movl    $mutex, %edi
    .loc 1 83 0
    subl    $1, %eax
    movl    %eax, 0(%rbp)
    .loc 1 84 0
    call    pthread_mutex_unlock
```

1.2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

1. Η εκδοχή του προγράμματος με σημαφόρους είναι η εξής:

```

/*
 * 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 <errno.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)

int NTHREADS=3;
sem_t *sem;

/*********************\
 * Compile-time parameters *
\********************/

/*
 * 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 *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 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;
    }
}

```

```

oslab25@orion: ~/exercise3
/*
 * 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) { //The void type is used when a function can
    receive arguments of any type, here arg is used to pass the thread number
    int line, color_val[x_chars];
    for(line=(intptr_t)arg; line<y_chars; line+=NTHREADS) { //i, i+nthreads, i+2nthreads,
    i+3nthreads etc until we reach the limit y_chars
        compute_mandel_line(line, color_val); //creating the line

        sem_wait(&sem[(intptr_t)arg]); //checks if semaphore associated with the current
        thread is 1 and if so it decreases and outputs the line, if not it waits till the semaphore
        is
        //released by another thread.

        output_mandel_line(1, color_val);
        sem_post(&sem[((intptr_t)arg + 1) % NTHREADS]); //the function signals the semaphore
        here associated with the next thread, so that the next thread can start computing and outputting
        its own line.
    }
    return NULL;
}

int main(int argc, char *argv) {
    if(argc==2) {
        NTHREADS=argv[1];
    }
    pthread_t t[NTHREADS]; //table to store the thread ids that pthread_create is creating
    int i,ret;
    sem = (sem_t*)safe_malloc(NTHREADS * sizeof(sem_t)); //creating a table of semaphores,
    we need a semaphore for each thread, so that we can choose which thread outputs its line next

    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;

    if(sem_init(&sem[0], 0, 1)<0) { //initialising the first semaphore:unlocked(1), so that
    at the first thread prints the first line
        perror("sem_init");
        exit(1);
    }
    for (i = 1; i < NTHREADS; i++) { //the rest of semaphores:locked(0),we choose which one to unlock according to what line must be printed next
        sem_init(&sem[i], 0, 0);
    }

    for (i = 0; i<NTHREADS; i++) { //creation of threads
        ret = pthread_create(&t[i], NULL, compute_and_output_mandel_line, (void*)(int
ptr_t)i); //1rst:thread id;3rd: pointer to function that the thread will execute;4th:pointer
        to any arguments required by the function
        if (ret) { //error
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < NTHREADS; i++) { //termination of threads
        ret = pthread_join(t[i], NULL);
        if (ret) {
            perror("pthread_join");
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < NTHREADS; i++) { //destroying semaphores
        sem_destroy(&sem[i]);
    }
    free(sem);
    reset_xterm_color(1);
    return 0;
}

```

2. Η εκδοχή με μεταβλητές συνθήκης είναι η εξής:

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)

int y_chars = 50;
int x_chars = 90;

double xmin = -1.0, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

double xstep;
double ystep;

int NTHREADS=3;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t *conds;

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 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);
    }
}
```

1,17

```
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);
    }
}
```

```

int next_line_to_output=0;

void *compute_and_output_mandel_line(void *arg) {
    int line,color_val[x_chars];
    for(line=(intptr_t)arg; line<y_chars; line+=NTHREADS) {
        compute_mandel_line(line, color_val);
        pthread_mutex_lock(&mutex); //locks the mutex
        while (line != next_line_to_output) {
            pthread_cond_wait(&conds[(intptr_t)arg], &mutex); //waits on a condition variable if the line number is greater or equal to y_chars (i.e. all the lines have been computed and outputted by the previous threads)
        }
        next_line_to_output++;
        output_mandel_line(1, color_val);
        pthread_cond_signal(&conds[((intptr_t)arg+1)%NTHREADS]); //wakes up the next thread to compute its line
        pthread_mutex_unlock(&mutex); //the waiting thread wakes up and re-acquires the mutex lock
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if(argc>1){
        NTHREADS=atoi(argv[1]);
    }
    int i, ret;
    pthread_t t[NTHREADS];
    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;
    conds = (pthread_cond_t*)safe_malloc(NTHREADS * sizeof(pthread_cond_t));
    for (i = 0; i < NTHREADS; i++) {
        pthread_cond_init(&conds[i], NULL);
    }
    for(i=0; i<NTHREADS; i++){
        ret = pthread_create(&t[i], NULL, compute_and_output_mandel_line, (void*)(intptr_t)i);
        if (ret) {
            perror_pthread(ret, "pthread_create");
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < NTHREADS; i++) { //terminating of threads
        ret = pthread_join(t[i], NULL);
        if (ret) {

```

136,10-17 89%

```

        }
        next_line_to_output++;
        output_mandel_line(1, color_val);
        pthread_cond_signal(&conds[((intptr_t)arg+1)%NTHREADS]); //wakes up the next
thread to compute its line
        pthread_mutex_unlock(&mutex); //the waiting thread wakes up and re-acquires
he mutex lock
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if(argc>1){
        NTHREADS=atoi(argv[1]);
    }
    int i, ret;
    pthread_t t[NTHREADS];
    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;
    conds = (pthread_cond_t*)safe_malloc(NTHREADS * sizeof(pthread_cond_t));
    for (i = 0; i < NTHREADS; i++) {
        pthread_cond_init(&conds[i], NULL);
    }
    for(i=0; i<NTHREADS; i++){
        ret = pthread_create(&t[i], NULL, compute_and_output_mandel_line, (void*)(in
ptr_t)i);
        if (ret) {
            perror_pthread(ret, "pthread_create");
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < NTHREADS; i++) { //terminating of threads
        ret = pthread_join(t[i], NULL);
        if (ret) {
            perror_pthread(ret, "pthread_join");
            exit(EXIT_FAILURE);
        }
    }
    for (i = 0; i < NTHREADS; i++) {
        pthread_cond_destroy(&conds[i]);
    }
    pthread_mutex_destroy(&mutex);
    reset_xterm_color(1);
    return 0;
}

```

147,1

Bot

Ερωτήσεις:

1. Χρειάζονται τόσοι σημαφόροι όσα και τα νήματα που χρησιμοποιούνται προκειμένου να επιτευχθεί ο συγχρονισμός. Αυτό συμβαίνει καθώς κάθε σημαφόρος πρέπει να έχει από ένα νήμα, ώστε να εκτυπωθούν οι γραμμές με τη σωστή σειρά.

2. Για την ολοκλήρωση του σειριακού και του παράλληλου προγράμματος με δυο νήματα υπολογισμού απαιτούνται οι παρακάτω χρόνοι:

- Χρόνος για σειριακό πρόγραμμα

real	0m2.339s
user	0m0.740s
sys	0m0.024s

- Χρόνος για παράλληλο πρόγραμμα με 2 νήματα

real	0m0.722s
user	0m0.760s
sys	0m0.016s

Όπως αναμενόταν, ο χρόνος του παράλληλου προγράμματος είναι σημαντικά μικρότερος.

3. Και πάλι χρειαστήκαμε τόσες μεταβλητές συνθήκης, όσα και τα νήματα που διεκπεραιώνουν τον συγχρονισμό του output. Αν χρησιμοποιηθεί μόνο 1 μεταβλητή συνθήκης, τότε στο critical section του compute_and_output_mandel_line() θα πρέπει να χρησιμοποιηθεί η εντολή pthread_cond_broadcast() αντί για την pthread_cond_signal(), προκειμένου να ενεργοποιηθούν όλα τα νήματα τα οποία “ελέγχει” η μοναδική condition variable.

4. Χρόνος για σειριακό πρόγραμμα

real	0m2.339s
user	0m0.740s
sys	0m0.024s

- Χρόνος για παράλληλο με condition variables:

real	0m1.162s
user	0m0.744s
sys	0m0.032s

Παρατηρούμε ότι το εκτελέσιμο στο οποίο εκτελείται συγχρονισμός εμφανίζει πράγματι επιτάχυνση. Αυτό συμβαίνει γιατί το computing Των προς εκτύπωση γραμμών βρίσκεται εκτός του κρίσιμου τμήματος των νημάτων και έτσι τα νήματα υπολογίζουν παράλληλα τις γραμμές που τους έχουν ανατεθεί. Ωστόσο, εάν ο υπολογισμός γινόταν εντός του Critical section, πιθανότατα θα είχαμε κάποια χρονική επιμήκυνση.

5. Εάν πατήσουμε Ctrl+C, ενώ το πρόγραμμά μας τρέχει, παρατηρούμε ότι το τερματικό αφήνεται στο χρώμα του τελευταίου χαρακτήρα που εκτυπώνεται. Για να το αποφύγουμε αυτό, θα μπορούσαμε να υλοποιήσουμε έναν signal-handler (αφού το Ctrl+C είναι ένα σήμα που στέλνουμε από το πληκτρολόγιο) που θα πιάνει αυτό το σήμα και θα εκτελεί την εντολή επαναφοράς χρώματος reset_xterm_color(1).

Κώδικας για τον handler:

```
struct sigstastion sa;
sa.sa_handler=sigint_handler;
sa.sa_flags=0;
sigemptyset(&sa.sa_mask);
if(sigaction(SIGINT, &sa, NULL)<0){
    perror("sigaction");
    exit(1);
}
void sigint_handler(int signum) {
    reset_xterm_color(1);
```

```
    exit(1);  
}
```