ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργία Συστήματα Υπολογιστών

Αναφορά για την 3η Εργαστηριακή Άσκηση: Συγχρονισμός

Ον/μο :Μεσσής Κωνσταντίνος
Α.Μ.: :03111122

Ον/μο :Μπερέτσος Θεόδωρος
Α.Μ.: :03111612

Ημερομηνία παράδοσης: 22/1/2015

3.1 Συγχρονισμός σε υπάρχοντα κώδικα

-Πηγαίος κώδικας (source code)

simplesync.c

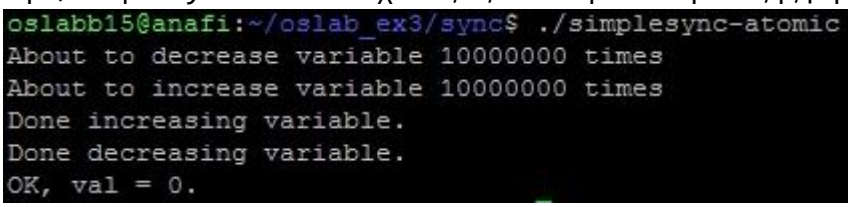
```
/*
 * simplesync.c
 *
 * A simple synchronization exercise.
 *
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 *
 */

#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 lock;

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

```
                        pthread_mutex_lock(&lock);
                        ++(*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) {
                        __sync_sub_and_fetch(ip,1);
                        /* You can modify the following line */
                        //--(*ip);
                        /* ... */
                } else {
                        /* ... */
                        /* You cannot modify the following line */
                        pthread_mutex_lock(&lock);
                        --(*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);
        }
```

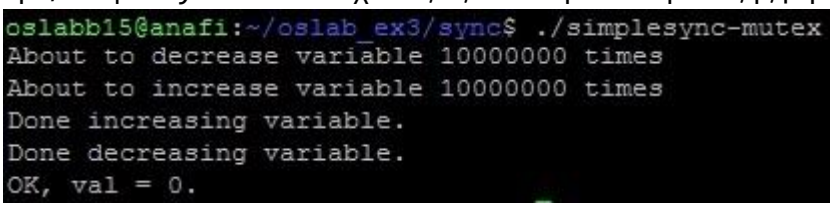
```
        /*
         * Wait for threads to terminate
         */
        ret = pthread_join(t1, NULL);
        if (ret)
                perror_pthread(ret, "pthread_join");
        ret = pthread_join(t2, NULL);
        if (ret)
                perror_pthread(ret, "pthread_join");

        /*
         * Is everything OK?
         */
        ok = (val == 0);

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

        return ok;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./simplesync-atomic` έχει ως έξοδο στην οθόνη το εξής μήνυμα:



Η εντολή `./simplesync-mutex` έχει ως έξοδο στην οθόνη το εξής μήνυμα:



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: 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;

Απάντηση:

Ο χρόνος εκτέλεσης του κώδικα χωρίς συγχρονισμό είναι αισθητά μικρότερος κι αυτό γιατί η επιβολή του συγχρονισμού χρειάζεται επιπρόσθετες εντολές που απαιτούν παραπάνω χρόνο.

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

Απάντηση:

Η μέθοδος συγχρονισμού με POSIX Mutexes είναι πιο αργή από αυτή των ατομικών λειτουργιών καθώς οι τελευταίες γίνονται σε επίπεδο υλικού άρα εκτελούνται και ταχύτερα.

Ερώτηση 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`.

Απάντηση:

Αν κρίνουμε από τα `.loc` σωστά η `__sync_add _and_fetch` μεταγλωττίζεται στις παρακάτω εντολές:

```
movq    -16(%rbp), %rax
lock addl       $1, (%rax)
```

Ερώτηση 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.

Απάντηση:

Αν κρίνουμε από τα `.loc` σωστά η `pthread_mutex_lock` μεταγλωττίζεται στις παρακάτω εντολές:

```
movl    $lock, %edi
call    pthread_mutex_lock
```

3.2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

-Πηγαίος κώδικας (source code)

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 <errno.h>
#include <semaphore.h>
#include <pthread.h>
```

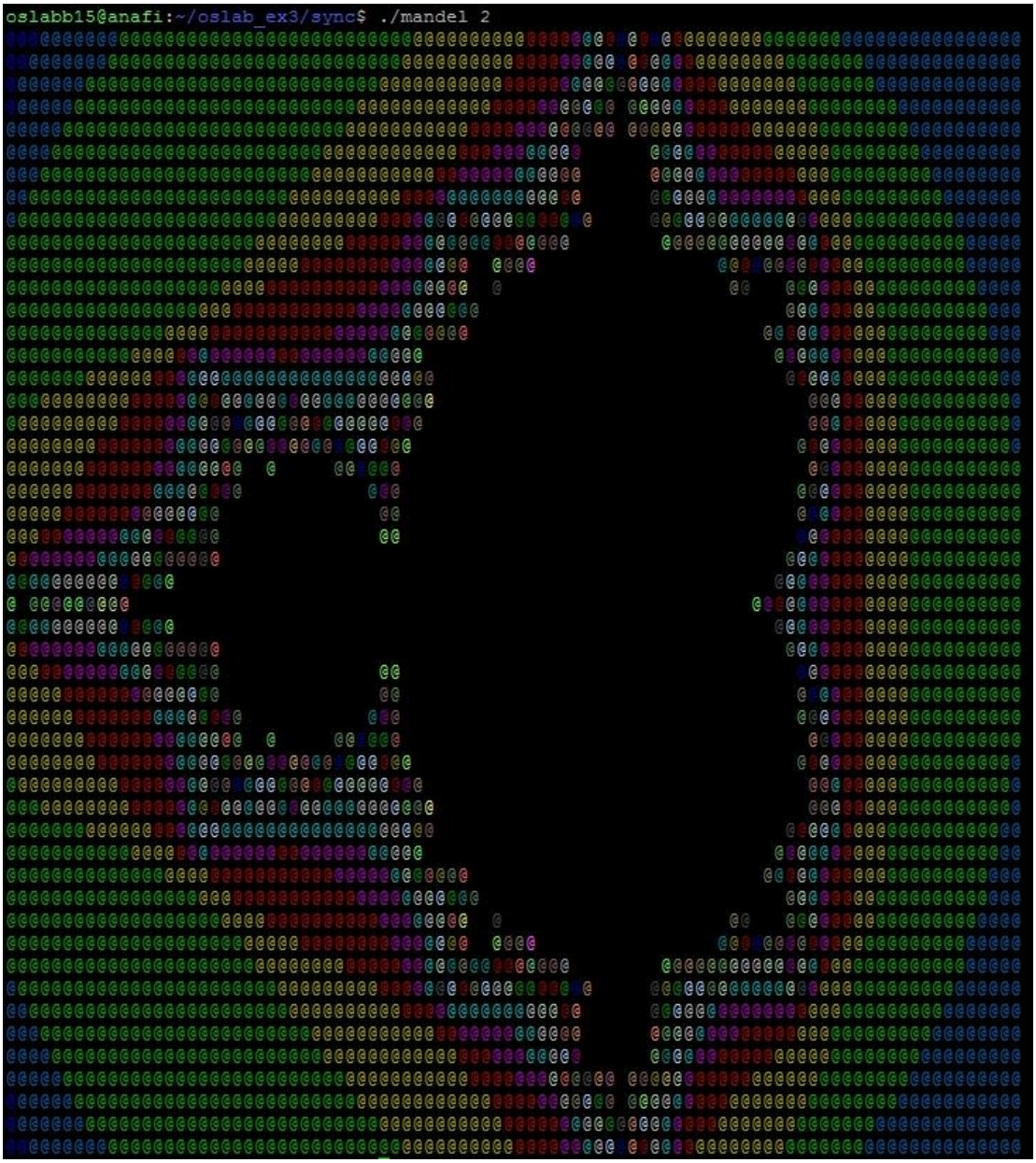
```c
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/


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

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

struct thread_info_struct{
       pthread_t tid;
       int id;
};

int NTHREADS;
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.
 */

if(argc != 2){
              printf("error 1 argument needed as number of threads used");
       exit(1);
}
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;
}
```

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

sem_t sem[NTHREADS];



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 i)
{
        /*
         * 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[i]);
        output_mandel_line(fd, color_val);
        sem_post(&sem[(i+1)%NTHREADS]);
}

void *thread_start_fn(void *arg){
        struct thread_info_struct *thr = arg;
        int i=thr->id,j;

        for(j=0;j<y_chars;j++){
                if ((j%NTHREADS) == i) compute_and_output_mandel_line(1,j,i);
        }
        return NULL;
}


        int i,ret;
        struct thread_info_struct thr[NTHREADS];

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

        sem_init(&sem[0],0,1);
        for(i=1;i<NTHREADS;i++){
                sem_init(&sem[i],0,0);
        }

        for(i=0;i<NTHREADS;i++){
                thr[i].id=i;
                ret=pthread_create(&thr[i].tid,NULL,thread_start_fn,&thr[i]);
                if(ret){
                        perror_pthread(ret,"pthread_create");
                        exit(1);
                }
        }

        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor '1', i.e., standard output.
         */
        for (i=0;i<NTHREADS;i++){
                ret = pthread_join(thr[i].tid,NULL);
                if (ret){
                        perror_pthread(ret,"pthread_join");
                        exit(1);
                }
        }

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

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./mandel 2` έχει ως έξοδο στην οθόνη το εξής μήνυμα:



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση 1: Πόσοι σημαφόροι χρειάζονται για το σχήμα συγχρονισμού που υλοποιείτε;

Απάντηση:

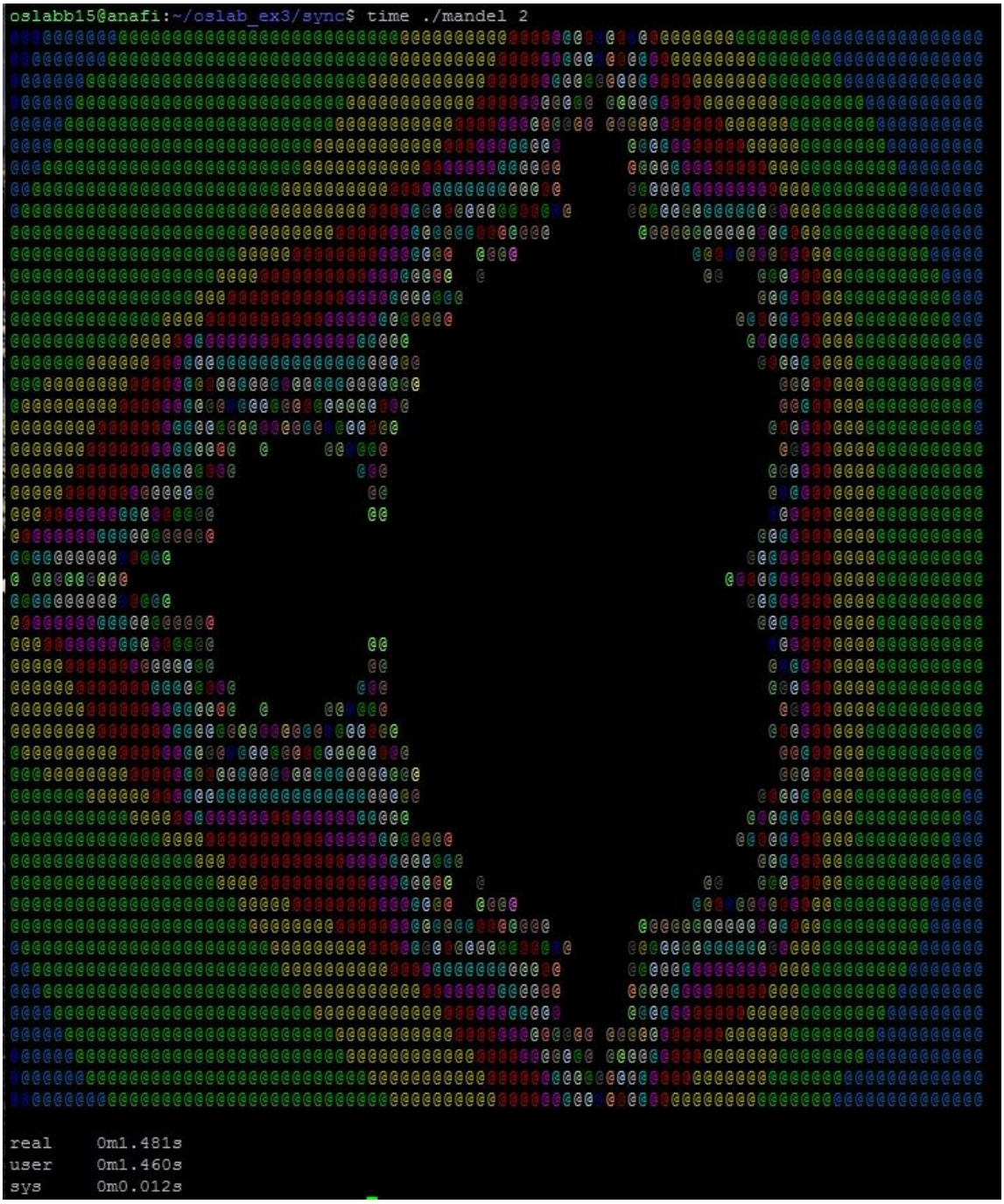
Στην υλοποίηση μας χρησιμοποιούμε το σύστημα κυκλικού κλειδώματος όπως αναφέρεται στο βιβλίο όπου κάθε νήμα έχει και τον δικό του σημαφόρο ώστε να τηρείται η σειριακή τους προτεραιότητα . Άρα μας χρειάζονται τόσοι σημαφόροι όσα τα νήματα ο αριθμός των οποίων δίνεται σαν είσοδος.

Ερώτηση 2: Πόσος χρόνος απαιτείται για την ολοκλήρωση του σειριακού και του παράλληλου προγράμματος με δύο νήματα υπολογισμού; Χρησιμοποιήστε την εντολή `time(1)` για να

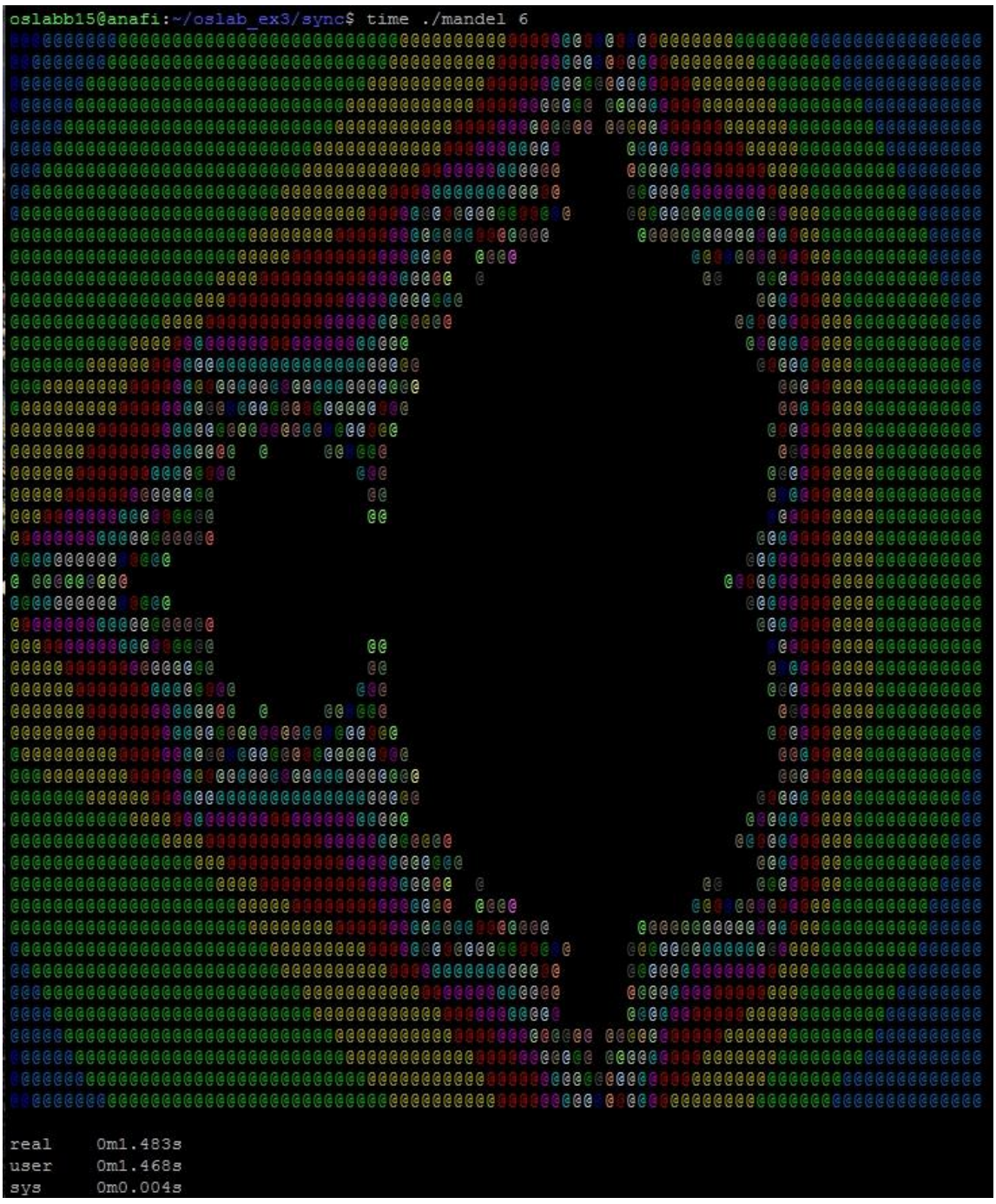
χρονομετρήσετε την εκτέλεση ενός προγράμματος, π.χ., `time sleep 2`. Για να έχει νόημα η μέτρηση, δοκιμάστε σε ένα μηχάνημα που διαθέτει επεξεργαστή δύο πυρήνων. Χρησιμοποιήστε την εντολή `cat /proc/cpuinfo` για να δείτε πόσους υπολογιστικούς πυρήνες διαθέτει κάποιο μηχάνημα.

Απάντηση:

Αν Με τη χρήση της `cat /proc/cpuinfo` παρατηρούμε ότι το μηχάνημά μας έχει 6 επεξεργαστικές μονάδες . Στο σύστημα αυτό με την εκτέλεση της `time ./mandel 2` παίρνουμε την εξής έξοδο:



Ενώ με την εκτέλεση της `time ./mandel 6`:



Ερώτηση 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;

Απάντηση:

Το πρόγραμμα δε χρειάζεται να περιέχει στο κρίσιμο τμήμα το κομμάτι του υπολογισμού παρά μόνο αυτό της τύπωσης αφού αυτό που μας νοιάζει είναι οι διαδοχικές τιμές του mandelbrot να τυπώνονται σειριακά (όχι να υπολογίζονται). Έτσι

στην υλοποίηση μας αφήνουμε τον υπολογισμό να γίνεται ασύγχρονα και περιλαμβάνουμε στο κρίσιμο τμήμα μόνο την τύπωση προκειμένου να το μειώσουμε.

Από εκεί και πέρα το πρόγραμμα που φτιάξαμε θα εμφανίζει επιτάχυνση κυρίως όταν ο αριθμός των νημάτων είναι ίδιος με το αριθμό των πυρήνων του μηχανήματος μας όπως φαίνεται και παραπάνω. Για μεγαλύτερο αριθμό νημάτων η ταχύτητα θα αρχίσει να μειώνεται καθώς θα πρέπει να εξυπηρετούνται περισσότερα νήματα από τον ίδιο αριθμό πυρήνων.

Ερώτηση 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;

Απάντηση:

Αν στο πρόγραμμά μας πατήσουμε `Ctrl-C` την ώρα που αυτό εκτελείται η έξοδος του mandelbrot δεν δίνεται ολόκληρη αλλά πέρα από αυτό τα χρώματα του τερματικού μας έχουν αλλάξει. Αυτό συμβαίνει γιατί δεν έχει εκτελεστεί η εντολή `reset_xterm_color(1)` που επαναφέρει τα χρώματα όπως ήταν πριν τα αλλάξει ο `mandel.c`. Για να μην γίνει αυτό αφού το πάτημα του `Ctrl-C` είναι ουσιαστικά η επιβολή ενός σήματος τερματισμού στο πρόγραμμα να βάλουμε κάποιον `signal handler` που θα εκτελεί την παραπάνω εντολή πριν τον τερματισμό της διεργασίας.

3.3 Επίλυση προβλήματος συγχρονισμού

-Πηγαίος κώδικας (source code)

kgarten.c

```
/*
 * kgarten.c
 *
 * A kindergarten simulator.
 * Bad things happen if teachers and children
 * are not synchronized properly.
 *
 *
 * Author:
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 *
 * Additional Authors:
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 * Anastassios Nanos <ananos@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 *
 */

#include <time.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
```

```c
/*
 * 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 virtual kindergarten */
struct kgarten_struct {

	/*
	 * Here you may define any mutexes / condition variables / other variables
	 * you may need.
	 */

	/* ... */

	/*
	 * You may NOT modify or use anything in the structure below this
	 * point. They are only meant to be used by the framework code,
	 * for verification.
	 */
	int vt;
	int vc;
	int ratio;

	pthread_mutex_t mutex;
	pthread_cond_t cond;
};

/*
 * 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 */

	struct kgarten_struct *kg;
	int is_child;  /* Nonzero if this thread simulates children, zero otherwise
*/

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

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 child_threads c_t_ratio\n\n"
		"Exactly three argument required:\n"
		"    thread_count: Total number of threads to create.\n"
		"    child_threads: The number of threads simulating children.\n"
		"    c_t_ratio: The allowed ratio of children to teachers.\n\n",
		argv0);
	exit(1);
}

void bad_thing(int thrid, int children, int teachers)
{
	int thing, sex;
	int namecnt, nameidx;
	char *name, *p;
	char buf[1024];

	char *things[] = {
		"Little %s put %s finger in the wall outlet and got electrocuted!",
		"Little %s fell off the slide and broke %s head!",
		"Little %s was playing with matches and lit %s hair on fire!",
		"Little %s drank a bottle of acid with %s lunch!",
		"Little %s caught %s hand in the paper shredder!",
		"Little %s wrestled with a stray dog and it bit %s finger off!"
	};
	char *boys[] = {
		"George", "John", "Nick", "Jim", "Constantine",
		"Chris", "Peter", "Paul", "Steve", "Billy", "Mike",
		"Vangelis", "Antony"
	};
	char *girls[] = {
		"Maria", "Irene", "Christina", "Helena", "Georgia", "Olga",
		"Sophie", "Joanna", "Zoe", "Catherine", "Marina", "Stella",
		"Vicky", "Jenny"
	};

	thing = rand() % 4;
	sex = rand() % 2;

	namecnt = sex ? sizeof(boys)/sizeof(boys[0]) :
sizeof(girls)/sizeof(girls[0]);
	nameidx = rand() % namecnt;
	name = sex ? boys[nameidx] : girls[nameidx];

	p = buf;
	p += sprintf(p, "*** Thread %d: Oh no! ", thrid);
	p += sprintf(p, things[thing], name, sex ? "his" : "her");
	p += sprintf(p, "\n*** Why were there only %d teachers for %d
children?!\n",
```

```
                teachers, children);

        /* Output everything in a single atomic call */
        printf("%s", buf);
}

void child_enter(struct thread_info_struct *thr)
{
        if (!thr->is_child) {
                fprintf(stderr, "Internal error: %s called for a Teacher thread.\n",
                        __func__);
                exit(1);
        }

        fprintf(stderr, "THREAD %d: CHILD ENTER\n", thr->thrid);

        pthread_mutex_lock(&thr->kg->mutex);
        while (thr->kg->vc >= thr->kg->vt * thr->kg->ratio) pthread_cond_wait(&thr-
>kg->cond,&thr->kg->mutex);
        ++(thr->kg->vc);
        pthread_cond_broadcast(&thr->kg->cond);
        pthread_mutex_unlock(&thr->kg->mutex);
}

void child_exit(struct thread_info_struct *thr)
{

        if (!thr->is_child) {
                fprintf(stderr, "Internal error: %s called for a Teacher thread.\n",
                        __func__);
                exit(1);
        }

        fprintf(stderr, "THREAD %d: CHILD EXIT\n", thr->thrid);

        pthread_mutex_lock(&thr->kg->mutex);
        --(thr->kg->vc);
        pthread_cond_broadcast(&thr->kg->cond);
        pthread_mutex_unlock(&thr->kg->mutex);
}

void teacher_enter(struct thread_info_struct *thr)
{
        if (thr->is_child) {
                fprintf(stderr, "Internal error: %s called for a Child thread.\n",
                        __func__);
                exit(1);
        }

        fprintf(stderr, "THREAD %d: TEACHER ENTER\n", thr->thrid);

        pthread_mutex_lock(&thr->kg->mutex);
        ++(thr->kg->vt);
        pthread_cond_broadcast(&thr->kg->cond);
        pthread_mutex_unlock(&thr->kg->mutex);
}

void teacher_exit(struct thread_info_struct *thr)
{
        if (thr->is_child) {
                fprintf(stderr, "Internal error: %s called for a Child thread.\n",
                        __func__);
```

```
                exit(1);
        }

        fprintf(stderr, "THREAD %d: TEACHER EXIT\n", thr->thrid);

        pthread_mutex_lock(&thr->kg->mutex);
        while(thr->kg->vc >= (thr->kg->vt-1) * thr->kg->ratio)
pthread_cond_wait(&thr->kg->cond,&thr->kg->mutex);
        --(thr->kg->vt);
        pthread_cond_broadcast(&thr->kg->cond);
        pthread_mutex_unlock(&thr->kg->mutex);
}

/*
 * Verify the state of the kindergarten.
 */
void verify(struct thread_info_struct *thr)
{
         struct kgarten_struct *kg = thr->kg;
         int t, c, r;

         c = kg->vc;
         t = kg->vt;
         r = kg->ratio;

         fprintf(stderr, "             Thread %d: Teachers: %d, Children: %d\n",
                 thr->thrid, t, c);

         if (c > t * r) {
                 bad_thing(thr->thrid, c, t);
                 exit(1);
         }
}


/*
 * A single thread.
 * It simulates either a teacher, or a child.
 */
void *thread_start_fn(void *arg)
{
        /* We know arg points to an instance of thread_info_struct */
        struct thread_info_struct *thr = arg;
        char *nstr;

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

        nstr = thr->is_child ? "Child" : "Teacher";
        for (;;) {
                fprintf(stderr, "Thread %d [%s]: Entering.\n", thr->thrid, nstr);
                if (thr->is_child)
                        child_enter(thr);
                else
                        teacher_enter(thr);

                fprintf(stderr, "Thread %d [%s]: Entered.\n", thr->thrid, nstr);

                /*
                 * We're inside the critical section,
                 * just sleep for a while.
                 */
```

```
                /* usleep(rand_r(&thr->rseed) % 1000000 / (thr->is_child ? 10000 :
1)); */
                pthread_mutex_lock(&thr->kg->mutex);
                verify(thr);
                pthread_mutex_unlock(&thr->kg->mutex);

                usleep(rand_r(&thr->rseed) % 1000000);

                fprintf(stderr, "Thread %d [%s]: Exiting.\n", thr->thrid, nstr);
                /* CRITICAL SECTION END */

                if (thr->is_child)
                        child_exit(thr);
                else
                        teacher_exit(thr);

                fprintf(stderr, "Thread %d [%s]: Exited.\n", thr->thrid, nstr);

                /* Sleep for a while before re-entering */
                /* usleep(rand_r(&thr->rseed) % 100000 * (thr->is_child ? 100 : 1));
*/
                usleep(rand_r(&thr->rseed) % 100000);

                pthread_mutex_lock(&thr->kg->mutex);
                verify(thr);
                pthread_mutex_unlock(&thr->kg->mutex);
        }

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

        return NULL;
}


int main(int argc, char *argv[])
{
        int i, ret, thrcnt, chldcnt, ratio;
        struct thread_info_struct *thr;
        struct kgarten_struct *kg;

        /*
         * Parse the command line
         */
        if (argc != 4)
                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], &chldcnt) < 0 || chldcnt < 0 || chldcnt > thrcnt) {
                fprintf(stderr, "`%s' is not valid for `child_threads'\n", argv[2]);
                exit(1);
        }
        if (safe_atoi(argv[3], &ratio) < 0 || ratio < 1) {
                fprintf(stderr, "`%s' is not valid for `c_t_ratio'\n", argv[3]);
                exit(1);
        }


        /*
         * Initialize kindergarten and random number generator
         */
```

```
        srand(time(NULL));

        kg = safe_malloc(sizeof(*kg));
        kg->vt = kg->vc = 0;
        kg->ratio = ratio;

        ret = pthread_mutex_init(&kg->mutex, NULL);
        if (ret) {
                perror_pthread(ret, "pthread_mutex_init");
                exit(1);
        }

        /* ... */

        /*
         * Create threads
         */
        thr = safe_malloc(thrcnt * sizeof(*thr));

        for (i = 0; i < thrcnt; i++) {
                /* Initialize per-thread structure */
                thr[i].kg = kg;
                thr[i].thrid = i;
                thr[i].thrcnt = thrcnt;
                thr[i].is_child = (i < chldcnt);
                thr[i].rseed = rand();

                /* 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);
                }
        }

        printf("OK.\n");

        return 0;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./kgarten 10 7 2` μια τυχαία χρονική στιγμή που το τερματίσουμε έχει ως έξοδο στην οθόνη το εξής μήνυμα:

```
oslabb15@anafi:~/oslab_ex3/sync$ ./kgarten 10 7 2
Thread 1 of 10. START.
Thread 1 [Child]: Entering.
THREAD 1: CHILD ENTER
Thread 2 of 10. START.
Thread 2 [Child]: Entering.
THREAD 2: CHILD ENTER
Thread 3 of 10. START.
Thread 3 [Child]: Entering.
THREAD 3: CHILD ENTER
Thread 4 of 10. START.
Thread 4 [Child]: Entering.
THREAD 4: CHILD ENTER
Thread 5 of 10. START.
Thread 5 [Child]: Entering.
THREAD 5: CHILD ENTER
Thread 6 of 10. START.
Thread 6 [Child]: Entering.
THREAD 6: CHILD ENTER
Thread 7 of 10. START.
Thread 7 [Teacher]: Entering.
THREAD 7: TEACHER ENTER
Thread 7 [Teacher]: Entered.
            Thread 7: Teachers: 1, Children: 0
Thread 1 [Child]: Entered.
            Thread 1: Teachers: 1, Children: 1
Thread 2 [Child]: Entered.
            Thread 2: Teachers: 1, Children: 2
Thread 8 of 10. START.
Thread 8 [Teacher]: Entering.
THREAD 8: TEACHER ENTER
Thread 8 [Teacher]: Entered.
            Thread 8: Teachers: 2, Children: 2
Thread 3 [Child]: Entered.
            Thread 3: Teachers: 2, Children: 3
Thread 4 [Child]: Entered.
```

```
            Thread 4: Teachers: 2, Children: 4
Thread 9 of 10. START.
Thread 9 [Teacher]: Entering.
THREAD 9: TEACHER ENTER
Thread 9 [Teacher]: Entered.
            Thread 9: Teachers: 3, Children: 4
Thread 5 [Child]: Entered.
            Thread 5: Teachers: 3, Children: 5
Thread 6 [Child]: Entered.
            Thread 6: Teachers: 3, Children: 6
Thread 0 of 10. START.
Thread 0 [Child]: Entering.
THREAD 0: CHILD ENTER
Thread 6 [Child]: Exiting.
THREAD 6: CHILD EXIT
Thread 6 [Child]: Exited.
Thread 0 [Child]: Entered.
            Thread 0: Teachers: 3, Children: 6
Thread 1 [Child]: Exiting.
THREAD 1: CHILD EXIT
Thread 1 [Child]: Exited.
            Thread 6: Teachers: 3, Children: 5
Thread 6 [Child]: Entering.
THREAD 6: CHILD ENTER
Thread 6 [Child]: Entered.
            Thread 6: Teachers: 3, Children: 6
            Thread 1: Teachers: 3, Children: 6
Thread 1 [Child]: Entering.
THREAD 1: CHILD ENTER
Thread 0 [Child]: Exiting.
THREAD 0: CHILD EXIT
Thread 0 [Child]: Exited.
Thread 1 [Child]: Entered.
            Thread 1: Teachers: 3, Children: 6
Thread 3 [Child]: Exiting.
THREAD 3: CHILD EXIT
Thread 3 [Child]: Exited.
            Thread 0: Teachers: 3, Children: 5
Thread 0 [Child]: Entering.
THREAD 0: CHILD ENTER
Thread 0 [Child]: Entered.
            Thread 0: Teachers: 3, Children: 6
            Thread 3: Teachers: 3, Children: 6
Thread 3 [Child]: Entering.
THREAD 3: CHILD ENTER
Thread 7 [Teacher]: Exiting.
THREAD 7: TEACHER EXIT
Thread 1 [Child]: Exiting.
THREAD 1: CHILD EXIT
Thread 1 [Child]: Exited.
Thread 3 [Child]: Entered.
            Thread 3: Teachers: 3, Children: 6
Thread 5 [Child]: Exiting.
```

-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: Έστω ότι ένας από τους δασκάλους έχει αποφασίσει να φύγει, αλλά δεν μπορεί ακόμη να το κάνει καθώς περιμένει να μειωθεί ο αριθμός των παιδιών στο χώρο (κρίσιμο τμήμα). Τι συμβαίνει στο σχήμα συγχρονισμού σας για τα νέα παιδιά που καταφτάνουν και επιχειρούν να μπουν στο χώρο;

Απάντηση:

Στο σύστημα συγχρονισμού μας στην έξοδο κάθε δασκάλου και στην είσοδο κάθε παιδιού ελέγχουμε επαναληπτικά αν η ενέργεια αυτή και η αλλαγή των μεταβλητών vt και vc θα δημιουργήσει μετέπειτα πρόβλημα . Αν ναι, με μια conditional variable μπλοκάρουμε κάθε νήμα και το ξεμπλοκάρουμε με broadcast στην είσοδο ενός δασκάλου ή την έξοδο ενός παιδιού έτσι ώστε να ελέγξει εκ νέου αν η επερχόμενη αλλαγή συνεχίζει να μας βλάπτει. Αν όχι το 1ο νήμα εξυπηρετείται αλλιώς η διαδικασία ακολουθεί την παραπάνω αλληλουχία. Στο ζήτημα που θέτει η άσκηση αν η έξοδος του δασκάλου δημιουργεί πρόβλημα τότε το νήμα του μπλοκάρεται και δεν μπαίνει στο κρίσιμο τμήμα μέχρι να δημιουργηθούν οι συνθήκες (έξοδοι παιδιών είσοδοι νέων καθηγητών) ώστε η έξοδός του να μην προκαλεί πρόβλημα. Τα νήματα παιδιών που έρχονται μετέπειτα μπλοκάρονται και αυτά μέχρις ότου να μπει στο κρίσιμο τμήμα του ο καθηγητής που θέλει να φύγει και δεν θα μπουν στο δικό τους κρίσιμο τμήμα αν αντίστοιχα δεν δημιουργηθούν οι κατάλληλες συνθήκες

Ερώτηση2: Περιγράψτε καταστάσεις συναγωνισμού (races) που προκύπτουν στον κώδικα του kgarten.c που επιχειρεί να επαληθεύσει την ορθότητα του σχήματος συγχρονισμού που υλοποιείτε.

Απάντηση:

Στο πρόγραμμα καταστάσεις συναγωνισμού δημιουργούνται σε προσπάθεια εισόδου ενός νήματος παιδιού ή εξόδου ενός νήματος δασκάλου που μπορούν να μας προκαλέσουν πρόβλημα. Ο έλεγχος για την συνέχιση της διαδικασίας που σχετίζεται με τη συνθήκη αλλαγής των τιμών των vc ,vt γίνεται επαναληπτικά και πρέπει να χρησιμοποιείται ο ίδιος mutex σε κάθε έλεγχο ανεξαρτήτως νήματος παιδιού ή δασκάλου για να ελέγχονται σωστά και με τη σειρά που πρέπει οι συνθήκες.

-Διαδικασία μεταγλώττισης και σύνδεσης

Αλλάξαμε κατάλληλα το Makefile που μας δίνετε ώστε να δημιουργεί τα εκτελέσιμα όλες τις ασκήσεις όπως φαίνεται παρακάτω:

```
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 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


## 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-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)

clean:
	rm -f *.s *.o pthread-test kgarten mandel
```