Λειτουργικά Συστήματα Υπολογιστών

Αναφορά - 3η Εργαστηριακή Άσκηση

Ομάδα: oslab04

Κατσιάνης Κωνσταντίνος - 03120183

Κουρής Γεώργιος - 03120116

6ο Εξάμηνο - Μάιος 2023

1.1 Συγχρονισμός σε υπάρχοντα κώδικα

Κάνουμε μεταγλώττιση των επιθυμητών προγραμμάτων.

```
oslab04@orion:~/ex3/sync_backup$ make
gcc -Wall -O2 -pthread -c -o pthread-test.o pthread-test.c
gcc -Wall -O2 -pthread -o pthread-test pthread-test.o
gcc -Wall -O2 -pthread -DSYNC_MUTEX -c -o simplesync-mutex.o simplesync.c
gcc -Wall -O2 -pthread -o simplesync-mutex simplesync-mutex.o
gcc -Wall -O2 -pthread -DSYNC_ATOMIC -c -o simplesync-atomic.o simplesync.c
gcc -Wall -O2 -pthread -o simplesync-atomic simplesync-atomic.o
gcc -Wall -O2 -pthread -c -o kgarten.o kgarten.c
gcc -Wall -O2 -pthread -o kgarten kgarten.o
gcc -Wall -O2 -pthread -c -o mandel-lib.o mandel-lib.c
gcc -Wall -O2 -pthread -c -o mandel.o mandel.c
gcc -Wall -O2 -pthread -o mandel mandel-lib.o mandel.o
oslab04@orion:~/ex3/sync_backup$
```

Στη συνέχεια, τρέχουμε τα εκτελέσιμα simplesync-mutex και simplesync-atomic που προέκυψαν (θα εξηγήσουμε στη συνέχεια πώς) από το simplesync.c:

```
oslab04@orion:~/ex3/sync_backup$ ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
NOT OK, val = -1303822.
oslab04@orion:~/ex3/sync_backup$
```

```
oslab04@orion:~/ex3/sync_backup$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -2176731.
oslab04@orion:~/ex3/sync_backup$
```

Αρχικά, παρατηρούμε ότι το αποτέλεσμα είναι διάφορο του 0, αλλά και ότι αν εκτελέσουμε ξανά τα προγράμματα, η έξοδος θα είναι διαφορετική. Αυτό οφείλεται στο ότι ο επεξεργαστής δεσμεύεται εναλλάξ από καθένα από τα δύο νήματα (που εκτελούν τις λειτουργίες αύξησης και μείωσης N φορές της μεταβλητής val κατά 1). Έτσι, η πρόσθεση/αφαίρεση κατά 1 είναι ατελής.

Στη συνέχεια θα εξηγήσουμε πώς παράγονται τα δύο εκτελέσιμα, παρατηρώντας το 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 simplesync-mutex simplesync-atomic kgarten mandel

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

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

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

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

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

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

Όπως αναφέρθηκε, δημιουργούνται 2 εκτελέσιμα, το simplesync-mutex και το simplesync-atomic. Αυτό οφείλεται στα flags -DSYNC_MUTEX και -DSYNC_ATOMIC. Συγκεκριμένα, μέσω του USE_ATOMIC_OPS στο αρχικό αρχείο simplesync.c, επιλέγουμε αν θα ακολουθήσουμε σχήμα συγχρονισμού με atomic operations (τιμή 1) ή σχήμα με posix mutexes (τιμή 0).

```
/* ... */
#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
```

```
if (USE_ATOMIC_OPS) {
        /* ... */
        __sync_fetch_and_add(&ip,1);

        /* ... */
} else {
        /*      */
```

Ο κώδικας:

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define perror_pthread(ret, msg) \
      do { errno = ret; perror(msg); } while (0)
#define N 10000000
#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 mutex0 = PTHREAD_MUTEX_INITIALIZER;
void *increase_fn(void *arg)
{
     int i,ret;
     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_fetch_and_add(&ip,1);

                /* ... */
```

```
            } else {
                /* ... */
                ret = pthread_mutex_lock(&mutex0);
                if(ret) {
                    perror_pthread(ret, "pthread_mutex lock error");
                    exit(1);
                }
                /* You cannot modify the following line */
                ++(*ip);
                ret = pthread_mutex_unlock(&mutex0);
                if(ret) {
                    perror_pthread(ret, "pthread_mutex unlock error");
                    exit(1);
                }
                /* ... */
            }
        }
        fprintf(stderr, "Done increasing variable.\n");

        return NULL;
}
void *decrease_fn(void *arg)
{
        int i,ret;
        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_fetch_and_add(&ip,-1);
                /* ... */
            } else {
                /* ... */
                ret = pthread_mutex_lock(&mutex0);
                if(ret) {
                    perror_pthread(ret, "pthread_mutex lock error");
                    exit(1);
                }
                /* You cannot modify the following line */
                --(*ip);
                ret = pthread_mutex_unlock(&mutex0);
                if(ret) {
                    perror_pthread(ret, "pthread_mutex unlock error");
                    exit(1);
                }
                /* ... */
            }
        }
        fprintf(stderr, "Done decreasing variable.\n");
        return NULL;
}
int main(int argc, char *argv[])
{
        int val, ret, ok;
        pthread_t t1, t2;
        val = 0;
        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;
}
```

Η νέα έξοδος του κάθε εκτελέσιμου, είναι σωστή (val=0):

```
oslab04@orion:~/ex3/sync$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
oslab04@orion:~/ex3/sync$
```

```
oslab04@orion:~/ex3/sync$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
oslab04@orion:~/ex3/sync$
```

Ερωτήματα:

1. Χωρίς συγχρονισμό, τα δύο εκτελέσιμα είναι τα ίδια (βέβαια η έξοδος διαφέρει από εκτέλεση σε εκτέλεση, όπως αναφέρθηκε παραπάνω):

   ```
   oslab04@orion:~/ex3/sync_backup$ time ./simplesync-atomic
   About to increase variable 10000000 times
   About to decrease variable 10000000 times
   Done decreasing variable.
   Done increasing variable.
   NOT OK, val = -734672.

   real    0m0.167s
   user    0m0.076s
   sys     0m0.000s
   oslab04@orion:~/ex3/sync_backup$
   ```

   Με συγχρονισμό, η έξοδος για καθένα από τα δύο εκτελέσιμα:

   ```
   oslab04@orion:~/ex3/sync$ time ./simplesync-mutex
   About to decrease variable 10000000 times
   About to increase variable 10000000 times
   Done decreasing variable.
   Done increasing variable.
   OK, val = 0.

   real    0m3.456s
   user    0ml.688s
   sys     0m0.016s
   oslab04@orion:~/ex3/sync$
   ```

```
oslab04@orion:~/ex3/sync$ time ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.

real    0m1.150s
user    0m0.620s
sys     0m0.000s
oslab04@orion:~/ex3/sync$
```

Παρατηρούμε ότι οι χρόνοι εκτέλεσης των προγραμμάτων με συγχρονισμό είναι σχετικά αυξημένη συγκριτικά με το αρχικό (χωρίς). Αυτό οφείλεται στο ότι στο πρόγραμμα χωρίς συγχρονισμό δεν ελέγχεται ποιο νήμα από τα δύο θα κάνει χρήση των υπολογιστικών πόρων του συστήματος. Αντίθετα, στα προγράμματα με συγχρονισμό κάθε νήμα εκτελεί ατομικά τον κώδικα στο κρίσιμο τμήμα που έχουμε ορίσει, οπότε η εκτέλεση αυτή δεν είναι παράλληλη, όπως ήταν προηγουμένως.

2. Εύκολα παρατηρούμε ότι η υλοποίηση σχήματος συγχρονισμού με atomic operations είναι πιο γρήγορη από ότι σχήματος με posix mutexes. Αυτό γίνεται διότι τα atomic operations χρησιμοποιούν υλοποίηση σε επίπεδο υλικού σε αντίθεση με τα posix mutexes που λεχουν πιο πολύπλοκη δομή και κάνουν χρήση περισσότερων εντολών σε επίπεδο assembly. Επίσης, τα posix mutexes χρησιμοποιούν συναρτήσεις sleep-wake για τις διεργασίες/νήματα, τα οποία προκαλούν καθυστερήσεις στην εκτέλεση του συγχρονισμού.

3. Εκτελέσαμε την παρακάτω εντολή για να παράξουμε το αρχείο κώδικα σε assembly.

   gcc -O2 -pthread -DSYNC_ATOMIC -S -g -o simplesync-atomic.s simplesync.c

   Τα τμήματα που μας ενδιαφέρουν (το πρώτο αφορά την αύξηση με το δεύτερο τη μείωση):

```
.L2:
        .loc 1 48 3 is_stmt 1 view .LVU11
        .loc 1 50 4 view .LVU12
        lock addq       $1, 8(%rsp)
        .loc 1 47 21 view .LVU13
.LVL4:
```

```
        .p2align 3
.L7:
        .loc 1 82 3 is_stmt 1 view .LVU31
        .loc 1 85 5 view .LVU32
        lock subq       $1, 8(%rsp)
        .loc 1 81 21 view .LVU33
.LVL11:
```

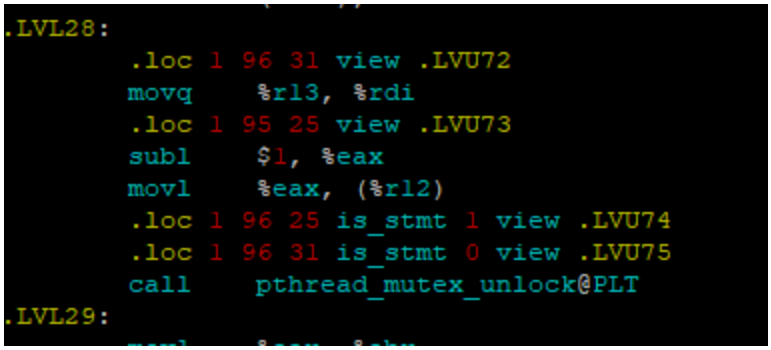
4. Για την αύξηση:

```
.L4:
        .loc 1 48 3 view .LVU15
        .loc 1 55 4 view .LVU16
        .loc 1 55 10 is_stmt 0 view .LVU17
        movq    %r13, %rdi
        call    pthread_mutex_lock@PLT
.LVL4:
```

```
LVL6:
        .loc 1 62 10 view .LVU22
        movq    %r13, %rdi
        .loc 1 61 4 view .LVU23
        addl    $1, %eax
        movl    %eax, (%r12)
        .loc 1 62 4 is_stmt 1 view .LVU24
        .loc 1 62 10 is_stmt 0 view .LVU25
        call    pthread_mutex_unlock@PLT
LVL7:
```

Για την μείωση:

```
        .p2align 3
.L13:
        .loc 1 82 3 view .LVU65
        .loc 1 89 4 view .LVU66
        .loc 1 89 10 is_stmt 0 view .LVU67
        movq    %r13, %rdi
        call    pthread_mutex_lock@PLT
.LVL26:
```


.LVL28:
.loc 1 96 31 view .LVU72
movq %r13, %rdi
.loc 1 95 25 view .LVU73
subl $1, %eax
movl %eax, (%r12)
.loc 1 96 25 is_stmt 1 view .LVU74
.loc 1 96 31 is_stmt 0 view .LVU75
call pthread_mutex_unlock@PLT
.LVL29:

1.2 Συγχρονισμός σε υπάρχοντα κώδικα

Ο κώδικας για την υλοποίηση με semaphores:

```
#include <stdint.h>
#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 <signal.h>
#include <errno.h>
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000
int y_chars = 50;
int x_chars = 90;
double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;
double xstep;
double ystep;
sem_t *semaphore;
int num_threads;
int safe_atoi(char *s, int *value) {
    long l;
    char *endp;
    l = strtol(s, &endp, 10);
    if(s != endp && *endp == '\0') {
```

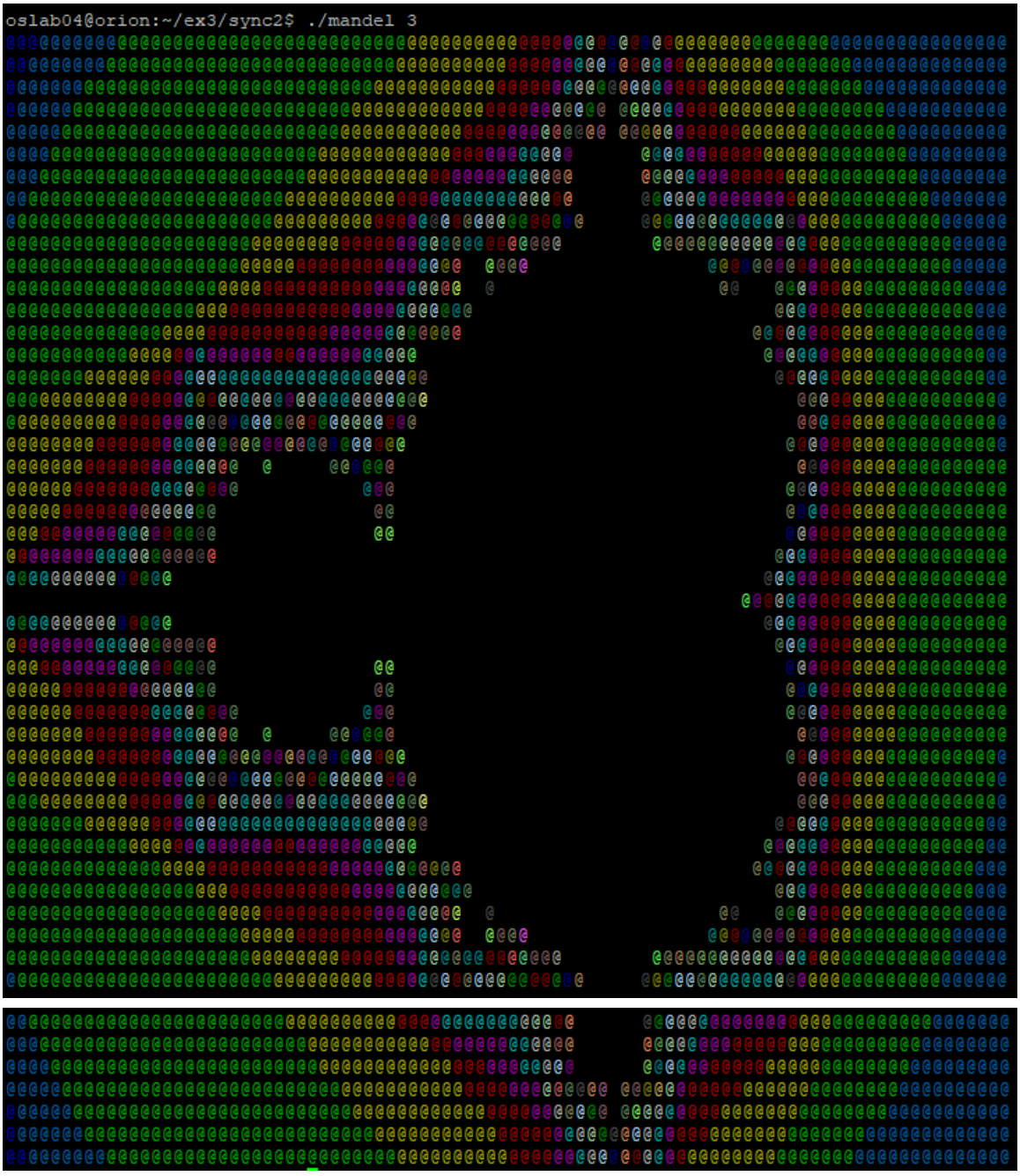
```
            *value=l;
            return 0;
        } else
            return -1;
}
void *safe_malloc(size_t size)
{
        void *p;
        if((p=malloc(size))==NULL) {
            fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n",
                size);
            exit(1);
        }
        return p;
}
void compute_mandel_line(int line, int color_val[])
{
           double x, y;
        int n;
        int val;
          y = ymax - ystep * 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;
            val = xterm_color(val);
            color_val[n] = val;
        }
```

```
}
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);
		}
	}

	if (write(fd, &newline, 1) != 1) {
		perror("compute_and_output_mandel_line: write newline");
		exit(1);
	}
}
void *compute_and_output_mandel_line(void *thr)
{
	int color_val[x_chars];
	int i;
	for(i=(int)thr; i<y_chars; i+=num_threads) {
		compute_mandel_line(i, color_val);
		if (sem_wait(&semaphore[i%num_threads]) < 0) {
			perror("sem_wait error");
			exit(1);
```

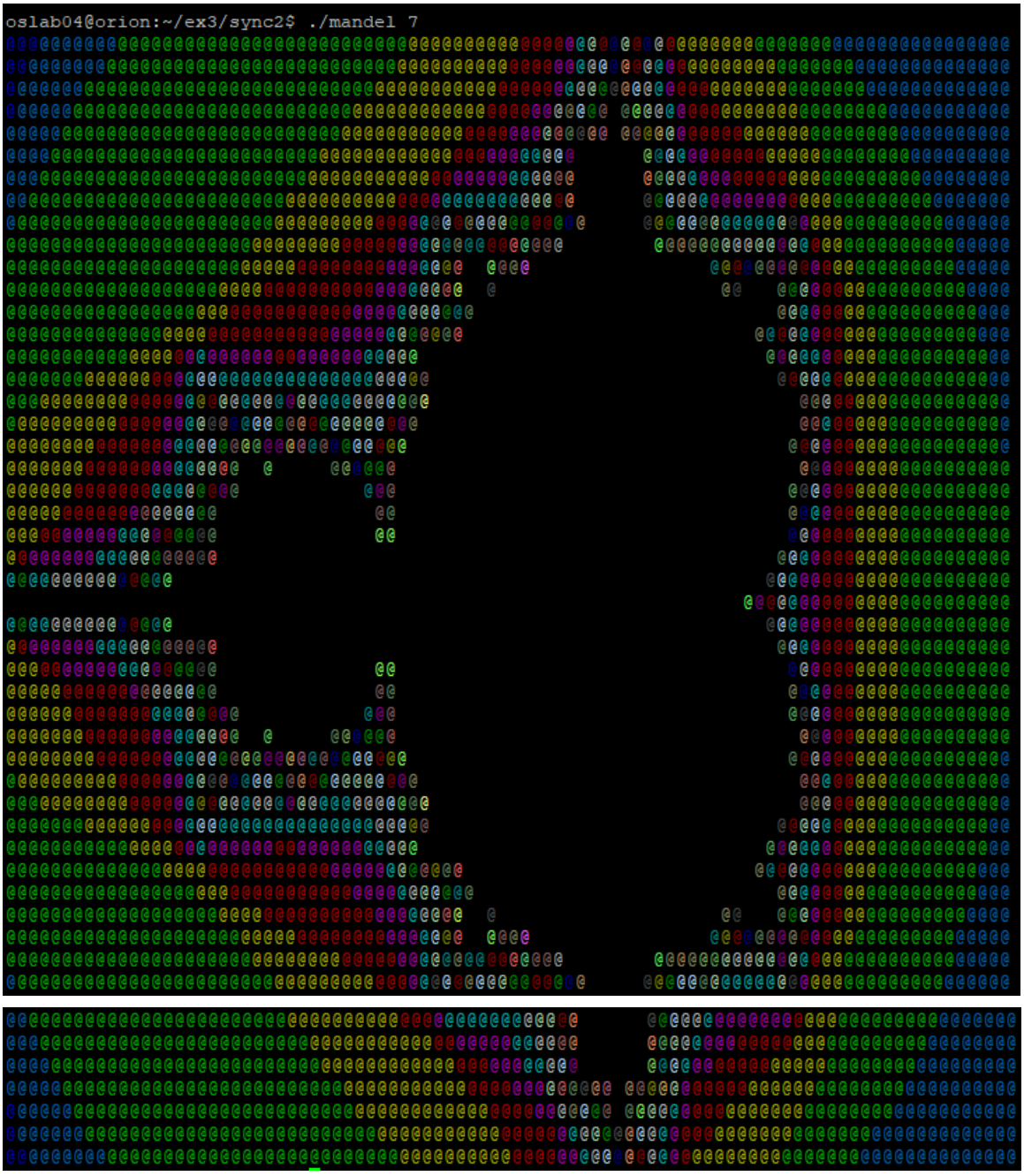
```
            }
            output_mandel_line(1, color_val);
            if (sem_post(&semaphore[(i+1)%num_threads]) < 0) {
                perror("sem_post error");
                exit(1);
            }
        }
        return NULL;
}
int main(int argc, char **argv)
{
        int i, ret;
        sigset_t sigset;
        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        if (argc != 2) {
            perror("Incorrect input, please insert only the number of threadsto create");
            exit(1);
        }
        if (safe_atoi(argv[1], &num_threads) < 0 || num_threads <= 0) {
            perror("input error");
            exit(1);
        }
        semaphore= (sem_t*)safe_malloc(num_threads*sizeof(sem_t));
        if (sem_init(&semaphore[0], 0, 1) < 0) {
            perror("sem_init error");
            exit(1);
        }
        for (i = 1; i < num_threads; i++) {
            if (sem_init(&semaphore[i], 0, 0) < 0) {
```

```
                perror("sem_init error");
                exit(1);
            }
        }
        pthread_t thread[num_threads];
        for (i = 0; i < num_threads; i++) {
            ret = pthread_create(&thread[i], NULL, compute_and_output_mandel_line,
(void*)(uintptr_t)i);
            if (ret) {
                perror("pthread_create error");
                exit(1);
            }
        }
        for (i = 0; i < num_threads; i++) {
            ret = pthread_join(thread[i], NULL);
            if (ret) {
                perror("pthread_join error");
            }
        }
        for (i = 0; i < num_threads; i++) {
            if (sem_destroy(&semaphore[i]) < 0) {
                perror("sem_destroy error");
                exit(1);
            }
        }
        reset_xterm_color(1);
        return 0;
}
```

Για num_threads = 3:



Για num_threads = 7:



Ο κώδικας για την υλοποίηση με condition variables:

```
#include <stdint.h>
#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 <signal.h>
#include <errno.h>
#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

int y_chars = 50;
int x_chars = 90;

double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

double xstep;
double ystep;

int num_threads;

pthread_cond_t cond[7];
```

```
void sigint_handler(int signum) {
	reset_xterm_color(1);
	exit(1);
}

int safe_atoi(char *s, int *value) {
	long l;
	char *endp;
	l = strtol(s, &endp, 10);
	if(s != endp && *endp == '\0') {
		*value=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 compute_mandel_line(int line, int color_val[])
{
	 double x, y;
```

```
        int n;
        int val;

        y = ymax - ystep * line;

        for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {

                val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
                if (val > 255)
                        val = 255;

                val = xterm_color(val);
                color_val[n] = val;
        }
}

void output_mandel_line(int fd, int color_val[])
{
        int i;

        char point ='@';
        char newline='\n';

        for (i = 0; i < x_chars; i++) {
                set_xterm_color(fd, color_val[i]);
                if (write(fd, &point, 1) != 1) {
                        perror("compute_and_output_mandel_line: write point");
                        exit(1);
                }
```

```
        }

        if (write(fd, &newline, 1) != 1) {
                perror("compute_and_output_mandel_line: write newline");
                exit(1);
        }
}
pthread_mutex_t condition_mutex[7] = {
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER,
PTHREAD_MUTEX_INITIALIZER
};

int first=1;
void *compute_and_output_mandel_line(void *thr)
{
        int color_val[x_chars];
        int i = (int)thr;
        while(1) {
                compute_mandel_line(i, color_val);
                pthread_mutex_lock(&condition_mutex[i%num_threads]);
                        printf("Waiting %d\n", (i)%num_threads);
                        printf("%d\n", i);

pthread_cond_wait(&cond[i%num_threads],&condition_mutex[i%num_threads]);
```

```
            pthread_mutex_unlock(&condition_mutex[i%num_threads]);
            printf("Output %d\n", (i)%num_threads);
            output_mandel_line(1, color_val);
            pthread_mutex_lock(&condition_mutex[(i+1)%num_threads]);

            printf("Waking %d\n", (i+1)%num_threads);
            if (pthread_cond_signal(&cond[(i+1)%num_threads]) < 0) {
                perror("sem_post error");
                exit(1);
            }
             pthread_mutex_unlock(&condition_mutex[(i+1)%num_threads]);

            i += num_threads;
            if (i >= y_chars)
                return NULL;
        }
        return NULL;
}

int main(int argc, char **argv)
{
        int i, ret;
        sigset_t sigset;

        xstep = (xmax - xmin) / x_chars;
        ystep = (ymax - ymin) / y_chars;
        if (argc != 2) {
            perror("Incorrect input, please insert only the number of threadsto create");
            exit(1);
        }
```

```
    if (safe_atoi(argv[1], &num_threads) < 0 || num_threads <= 0) {
        perror("input error");
        exit(1);
    }

    struct sigaction sa;
    sa.sa_handler = sigint_handler;
    sa.sa_flags = 0;
    sigemptyset(&sigset);
    sa.sa_mask = sigset;

    if (sigaction(SIGINT, &sa, NULL) < 0) {
        perror("sigaction");
        exit(1);
    }

    if (pthread_cond_init(&cond[0], NULL) < 0) {
         perror("pthread_cond_init error");
              exit(1);
    }
pthread_mutex_lock(&condition_mutex[0]);
    pthread_cond_signal(&cond[0]);
     pthread_mutex_unlock(&condition_mutex[0]);

    for (i = 1; i < num_threads; i++) {
        if (pthread_cond_init(&cond[i], NULL) < 0) {
            perror("pthread_cond_init error");
            exit(1);
        }
    }
```

```
    pthread_t thread[num_threads];
    for (i = 0; i < num_threads; i++) {
        ret = pthread_create(&thread[i], NULL, compute_and_output_mandel_line,
(void*)(uintptr_t)i);
        if (ret) {
            perror("pthread_create error");
            exit(1);
        }
    }
    printf("wake up first\n");

    for (i = 0; i < num_threads; i++) {
        ret = pthread_join(thread[i], NULL);
        if (ret) {
            perror("pthread_join error");
        }
    }
    for (i = 0; i < num_threads; i++) {
        if (pthread_cond_destroy(&cond[i]) < 0) {
            perror("pthread_cond_destroy error");
            exit(1);
        }
    }
    reset_xterm_color(1);
    return 0;
}
```

Ερωτήματα:

1. Ο αριθμός των σημαφόρων που χρειάζονται για το σχήμα συγχρονισμού είναι ίσος με τον αριθμό των threads.

2. Για την εκτέλεση της εντολής time ./mandel 1 (σειριακό) έχουμε:

```
real    0m1.471s
user    0m1.415s
sys     0m0.036s
```

   Και για την εκτέλεση της εντολής time ./mandel 2 (παράλληλο με 2 νήματα) έχουμε:

```
real    0m0.860s
user    0m1.261s
sys     0m0.040s
```

   Παρατηρούμε ότι ο χρόνος εκτέλεσης στην πρώτη περίπτωση είναι αρκετά μεγαλύτερος (περίπου διπλάσιος) από ότι στη δεύτερη με 2 νήματα.

   Μπορούμε να δούμε τον αριθμό πυρήνων αν εκτελέσουμε την εντολή τοπικά και δούμε τη γραμμή “cpu cores”.

3. Στη συγκεκριμένη εκδοχή χρησιμοποιούμε πλήθος condition variables ίσο με τον αριθμό των threads. Βέβαια, θα ήταν δυνατόν το πρόγραμμα να λειτουργήσει ακόμη και εάν είχαμε ένα μόνο condition variable, παρόλο που σε αυτήν την περίπτωση θα μπορούσαν να δημιουργηθούν προβλήματα.

4. Το παράλληλο πρόγραμμα που φτιάξαμε εμφανίζει όντως επιτάχυνση. Αυτό οφείλεται στο γεγονός ότι οι υπολογισμοί των γραμμών δεν βρίσκονται μέσα στο κρίσιμο τμήμα, είναι παράλληλοι και επομένως γίνονται μαζεμένα. Στη συνέχεια έχουμε την εκτύπωση, που απαιτεί τον συγχρονισμό των threads, για να εκτυπώνονται οι γραμμές με τον σωστό τρόπο (κρίσιμο κομμάτι).

5. Αν χρησιμοποιήσουμε Ctrl+C τότε η εκτέλεση θα διακοπεί χωρίς να γίνει reset το χρώμα, οπότε στη συνέχεια το χρώμα του τερματικού θα έχει αλλάξει. Για να το αλλάξουμε, αρκεί να κάνουμε reset στην περίπτωση που ο χρήστης πατήσει Ctrl+C. Ο κώδικας που προσθέσαμε είναι:

```c
/* case: usage of ctrl C*/
void sigint_handler(int signum) {
        reset_xterm_color(1);
        exit(1);
}
```

```c
struct sigaction sa;
sa.sa_handler = sigint_handler;
sa.sa_flags = 0;
sigemptyset(&sigset);
sa.sa_mask = sigset;
if (sigaction(SIGINT, &sa, NULL) < 0) {
        perror("sigaction");
        exit(1);
}
```