Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
6^{Ο} Εξάμηνο: Λειτουργικά Συστήματα 2022-23
Δανάη Σπέντζου (03120237)
Νεκτάριος Μπούμπαλος (03120441)

Άσκηση 3.1: Συγχρονισμός σε υπάρχοντα κώδικα

- Χρησιμοποιησούμε το παρεχόμενο Makefile για να μεταγλωττίσουμε και να τρέξουμε το πρόγραμμα.

```
oslab162@orion:~/exercise3/sync/sync$ make simplesync-atomic
gcc -Wall -O2 -pthread -DSYNC_ATOMIC -c -o simplesync-atomic.o simplesync.c
gcc -Wall -O2 -pthread -o simplesync-atomic simplesync-atomic.o
oslab162@orion:~/exercise3/sync/sync$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -3635912.
oslab162@orion:~/exercise3/sync/sync$ make simplesync-mutex
gcc -Wall -O2 -pthread -DSYNC_MUTEX -c -o simplesync-mutex.o simplesync.c
gcc -Wall -O2 -pthread -o simplesync-mutex simplesync-mutex.o
oslab162@orion:~/exercise3/sync/sync$ ./simplesync-mutex
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -90216.
```

Παρατηρούμε ότι το τελικό αποτέλεσμα της εκτέλεσης του simplesync δεν είναι 0. Αυτό συμβαίνει επειδή η εντολή πρόσθεσης ή αφαίρεσης στην assembly μεταφράζεται σε περισσότερες από μία εντολές. Έτσι, αν υπάρχει αλλαγή του νήματος που εκτελείται κατά τη διάρκεια αυτής της κρίσιμης περιοχής κώδικα, η λειτουργία δεν θα ολοκληρωθεί σωστά και τα αποτελέσματα θα είναι απρόβλεπτα. Επομένως, είναι απαραίτητος ο συγχρονισμός των δύο threads.

- Παρατηρούμε πώς παράγονται δύο διαφορετικά εκτελέσιμα simplesync-atomic, simplesync-mutex, από το ίδιο αρχείο πηγαίου κώδικα simplesync.c. Η διαφοροποίηση των εκτελέσιμων προέρχεται από την επιλογή που γίνεται στα ορίσματα SYNC_ATOMIC και SYNC_MUTEX:

```
#if defined(SYNC_ATOMIC) ^ defined(SYNC_MUTEX) == 0
# error You must #define exactly one of SYNC_ATOMIC or SYNC_MUTEX.
#endif
```

Αυτή η επιλογή καθορίζει ποιο μηχανισμό συγχρονισμού χρησιμοποιήσουμε, είτε atomic operations είτε mutex locks. Επίσης, βλέπουμε πως και στο Makefile τα ορίσματα αυτά προσδιορίζονται στην είσοδο ως arguments (DSYNC_ATOMIC, DSYNC_MUTEX):

```
## 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
```

```c
#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 mutex=PTHREAD_MUTEX_INITIALIZER;

void *increase_fn(void *arg)
{
        int ret;
        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_fetch_and_add(&ip, 1);
                        /* ... */
                } else {
                        ret=pthread_mutex_lock(&mutex);
                        if(ret)
                                perror_pthread(ret, "pthread_mutex_lock");
                        /* ... */
                        /* You cannot modify the following line */
                        ++(*ip);
                        /* ... */
                        ret=pthread_mutex_unlock(&mutex);
                        if(ret)
                                perror_pthread(ret, "pthread_mutex_unlock");
                }
        }
        fprintf(stderr, "Done increasing variable.\n");

        return NULL;
}

void *decrease_fn(void *arg)
{
        int ret;
        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_fetch_and_add(&ip,-1);
                } else {
                        ret=pthread_mutex_lock(&mutex);
                        if(ret)
                                perror_pthread(ret,"pthread_mutex_lock");
                        /* You cannot modify the following line */
                        --(*ip);
                        ret=pthread_mutex_unlock(&mutex);
                        if(ret)
                                perror_pthread(ret,"pthread_mutex_unlock");

                }
        }
        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);
        }
```

```c
	/*  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);

	ret=pthread_mutex_destroy(&mutex);
	if(ret)
		perror_pthread(ret,"pthread_mutex_destroy");


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

	return ok;
}
```

```
oslab162@orion:~/exercise3/sync$ ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
oslab162@orion:~/exercise3/sync$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
```

Ερωτήσεις:

1. Χρησιμοποιούμε την εντολή `time(1)` για να μετρήσουμε το χρόνο εκτέλεσης των εκτελέσιμων.

   Χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό:

```
oslab162@orion:~/exercise3/sync/sync$ time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 2903754.

real    0m0.306s
user    0m0.148s
sys     0m0.000s
oslab162@orion:~/exercise3/sync/sync$ time ./simplesync-mutex
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 3250816.

real    0m0.321s
user    0m0.156s
sys     0m0.000s
```

Χρόνος εκτέλεσης των εκτελέσιμων που εκτελούν συγχρονισμό:

```
oslab162@orion:~/exercise3/sync$ time ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.

real    0m1.201s
user    0m0.612s
sys     0m0.000s
oslab162@orion:~/exercise3/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    0m3.487s
user    0m1.684s
sys     0m0.016s
```

real : πραγματικός χρόνος που απαιτήθηκε για την εκτέλεση του προγράμματος

user : χρόνος που απαιτήθηκε για την εκτέλεση της διεργασίας από τη CPU, χωρίς να συμπεριλαμβάνονται οι εξωτερικές ενέργειες όπως η είσοδος/έξοδος

sys : χρόνος που απαιτήθηκε για εκτέλεση εντολών στο πυρήνα του λειτουργικού συστήματος κατά την εκτέλεση της διεργασίας

Ο χρόνος εκτέλεσης των εκτελέσιμων με συγχρονισμό είναι μεγαλύτερος σε σχέση με τον χρόνο εκτέλεσης του αρχικού προγράμματος. Αυτό συμβαίνει λόγω του overhead που εισάγεται από τις διαδικασίες συγχρονισμού που επιβάλλει πρόσθετες ενέργειες και ελέγχους που αυξάνουν τον χρόνο εκτέλεσης, όπως ο χρόνος που απαιτείται για τη λήψη και απελευθέρωση των locks ή στην εκτέλεση των atomic operations όταν υπάρχει ανταγωνισμός για την πρόσβαση στους κοινόχρηστους πόρους.

2. Η μέθοδος συγχρονισμού με atomic operations είναι γρηγορότερη σε σχέση με τη χρήση locks (mutex) για τον συγχρονισμό των threads καθώς έχουν χαμηλότερη πολυπλοκότητα, εκτελούνται απευθείας και εξοικονομούν τον χρόνο που θα απαιτούνταν για τη λήψη και απελευθέρωση των locks σε όλα τα threads.

3. Παράγοντας το κατάλληλο αρχείο simplesync-atomic.o που περιέχει τον assembly κώδικα που αντιστοιχεί στο πρόγραμμά μας:

    a. Για τη συνάρτηση increase_fn:

```
.L2:
        .loc 1 35 0
        lock addq       $1, 8(%rsp)
```

η οποία προσθέτει 1 στη μεταβλητή που βρίσκεται στη θέση 8(%rsp) χρησιμοποιώντας την εντολή lock για να εξασφαλιστεί η ατομικότητα της λειτουργίας.

    b. Για τη συνάρτηση decrease_fn:

```
.L7:
        .loc 1 65 0
        lock subq       $1, 8(%rsp)
```

η οποία αφαιρεί 1 από τη μεταβλητή που βρίσκεται στη θέση 8(%rsp) χρησιμοποιώντας την εντολή lock για να εξασφαλιστεί η ατομικότητα της λειτουργίας.

4. Παράγοντας το κατάλληλο αρχείο simplesync-mutex.o που περιέχει τον assembly κώδικα που αντιστοιχεί στο πρόγραμμά μας:
    a. Για τη συνάρτηση pthread_mutex_lock() :

```
.L2:
        .loc 1 38 0
        movl    $mutex, %edi
        call    pthread_mutex_lock
```

καλείται με την παράμετρο mutex που αντιστοιχεί στον καθορισμό του mutex που πρέπει να κλειδωθεί. Η εντολή movl $mutex, %edi φορτώνει την τιμή του mutex στον καταχωρητή %edi, και η εντολή call pthread_mutex_lock καλεί τη συνάρτηση pthread_mutex_lock για να εκτελέσει την κλείδωση του mutex.

    b. Για τη συνάρτηση pthread_mutex_unlock() :

```
.loc 1 45 0
call    pthread_mutex_unlock
```

Άσκηση 3.2: Παράλληλος υπολογισμός του συνόλου Mandelbrot

1η εκδοχή: Semaphores

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <signal.h>
#include <errno.h>
#include <pthread.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/***************************
 * Compile-time parameters *
 ***************************/
#define perror_pthread(ret,msg)\
        do{errno=ret;perror(msg);}while(0)
/*
 * 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 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 sigint_handler(int signum){
        reset_xterm_color(1);
        exit(1);
}


sem_t *sem;
int nrthreads;

void* compute_and_output_mandel_line(void *thread_index)
{
        /*
         * A temporary array, used to hold color values for the line being drawn
         */
        int i, color_val[x_chars];

        for(i=(int)thread_index;i<y_chars;i+=nrthreads){
                compute_mandel_line(i,color_val);
                if(sem_wait(&sem[(int)thread_index])<0){
                        perror("sem_wait");
                        exit(1);
                }
                output_mandel_line(1,color_val);
                if(sem_post(&sem[((int)thread_index+1)%nrthreads])<0){
                        perror("sem_post");
                        exit(1);
                }
        }
        return NULL;
}

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 * ptr = malloc(size);
        if(ptr==NULL){

                exit(EXIT_FAILURE);
        }
        return ptr;
}


int main(int argc,char **argv){
        int i,ret;
        int line;
```

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

        /*
         * draw the Mandelbrot Set, one line at a time.
         * Output is sent to file descriptor '1', i.e., standard output.
         */

        if(argc!=2)
                exit(1);

        nrthreads=atoi(argv[1]);

        /* sets up the signal handler for the SIGINT signal (Ctrl+C)*/
        struct sigaction sa;
        sa.sa_handler=sigint_handler;
        sa.sa_flags= 0;
        sigemptyset(&sa.sa_mask);

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

        /* the sem variable is a pointer to a semaphore array with the size of nrthreads
         * Semaphores are synchronization primitives used to control the synchronization between threads
         * when outputting lines of the Mandelbrot set to the terminal !
         * sem_wait(): locks a seaphore
         * sem_post(): releases the lock */

        sem=(sem_t*)malloc(nrthreads*sizeof(sem_t));

        /* the first semaphore allows the first thread to start immediately,
         * and the rest of the semaphores ensure that subsequent threads wait
         * until signaled by another thread before proceeding */

        if(sem_init(&sem[0],0,1)<0){
                perror("sem_init");
                exit(1);
        }

        for(i=1; i<nrthreads;i++){
                if(sem_init(&sem[i],0,0)<0){
                        perror("sem_init");
                        exit(1);
                }
        }

        pthread_t thread[nrthreads];
        for(i=0;i<nrthreads;i++){
                ret=pthread_create(&thread[i],NULL,compute_and_output_mandel_line,(void*)i);
                if(ret){
                        perror_pthread(ret, "pthread_create");
                        exit(1);
                }
        }


        /*Synchronization: By calling pthread_join, the main thread waits for each child thread to complete before proceeding.
         * This ensures that all the threads finish their work before the program exits. */
        for(i=0;i<nrthreads;i++){
                ret=pthread_join(thread[i],NULL);
                if(ret)
                        perror_pthread(ret, "pthread_join");
        }

        /*The purpose of this loop is to clean up and release the resources associated with the semaphores. */
        for(i=0;i<nrthreads;i++){
                if(sem_destroy(&sem[i])<0){
                                perror("sem_destroy");
                                exit(1);
                }
        }

        free(sem);


//      for (line = 0; line < y_chars; line++) {
//              compute_and_output_mandel_line(1, line);
//      }

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

```
oslab162@orion:~/exercise3/sync/sync$ ./mandel 10
```

2η εκδοχή: Conditions

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <signal.h>
#include <errno.h>
#include <pthread.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

/***************************
 *  * Compile-time parameters *
 *   ***************************/
#define perror_pthread(ret,msg) \
            do { errno = ret; perror(msg); } while (0)

/*
 *  * Output at the terminal 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;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
/*
 *  * 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;
}
```

```c
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.
 *    */
pthread_mutex_t output_mutex = PTHREAD_MUTEX_INITIALIZER;

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

pthread_mutex_t mutex;
pthread_cond_t cond;
int value = 0;
int nrthreads;
int current_line = 0;
int *current_thread;
struct custom_struct{
          int fd;
          int thrid;
          int thrcnt;
          pthread_t tid;
          int len;
          double *arr;
};

void *compute_and_output_mandel_line(void *arg){
        int color_val[x_chars];
        struct custom_struct *struct_1=arg;
        int thrid=struct_1->thrid;
        int fd=struct_1->fd;
        int thrcnt = struct_1->thrcnt;

        int line_x;
        for(line_x=thrid;line_x<y_chars;line_x+=thrcnt){
                pthread_mutex_lock(&mutex);
                while(*current_thread!=thrid)
                        pthread_cond_wait(&cond,&mutex);
                compute_mandel_line(line_x,color_val);
                output_mandel_line(fd,color_val);


                *current_thread=(*current_thread+1)%thrcnt;

                pthread_cond_broadcast(&cond);
                pthread_mutex_unlock(&mutex);
        }
        return NULL;
}

int main(int argc, char **argv){
        int i, ret;
        double *arr;

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

        if (argc != 2)
                exit(1);

        nrthreads = atoi(argv[1]);
        current_thread = &value;

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

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

```
        pthread_mutex_init(&mutex, NULL);
        pthread_cond_init(&cond, NULL);

        arr = safe_malloc(x_chars * sizeof(*arr));
        for (i = 0; i < x_chars; i++)
                arr[i] = 0;

        struct custom_struct *thread_index;
        pthread_t thread[nrthreads];
        thread_index = safe_malloc(nrthreads * sizeof(*thread_index));

        for (i = 0; i < nrthreads; i++) {
                thread_index[i].arr = arr;
                thread_index[i].len = 90;
                thread_index[i].thrid = i;
                thread_index[i].thrcnt = nrthreads;

                ret = pthread_create(&thread[i], NULL, compute_and_output_mandel_line, &thread_index[i]);
                if (ret) {
                        perror_pthread(ret, "pthread_create");
                        exit(1);
                }
        }
        for (i = 0; i < nrthreads; i++) {
                ret = pthread_join(thread[i], NULL);
                if (ret)
                        perror_pthread(ret, "pthread_join");
        }

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

```
oslab162@orion:~/exercise3/sync$ time ./mandel_condition 2
```

```
real    0m13.584s
user    0m4.456s
sys     0m0.068s
```

Ερωτήσεις:

1. Από τον παραπάνω κώδικα βλέπουμε ότι έχουμε κάνει χρήση τόσων σημαφόρων όσο είναι το πλήθος των νημάτων.
2. Στην 1η εκδοχή η εκτέλεση του προγράμματος με δύο νήματα πραγματοποιείται σε χρόνο:

   ```
   real    0m2.829s
   user    0m1.484s
   sys     0m0.020s
   ```

   Στην 2η εκδοχή η εκτέλεση του προγράμματος με δύο νήματα πραγματοποιείται σε χρόνο:

   ```
   real    0m13.584s
   user    0m4.456s
   sys     0m0.068s
   ```

3. Στη δεύτερη εκδοχή του προγράμματος χρησιμοποιήθηκε μία μεταβλητή συνθήκης (pthread_cond_t). Η μεταβλητή current_thread χρησιμοποιείται για να καθοριστεί ποιο νήμα είναι υπεύθυνο για τον υπολογισμό και την έξοδο της κάθε γραμμής και όσο το current_thread δεν είναι ισοδύναμο με το third το σύστημα βρίσκεται σε αναμονή. Το πρόβλημα επίδοσης που προκύπτει από τη χρήση μιας μόνο μεταβλητής για τον συγχρονισμό είναι ότι όλα τα νήματα αναμένουν στην ίδια μεταβλητή κατάστασης και το σήμα εκπομπής που στέλνεται από ένα νήμα τα ξυπνάει όλα. Αυτό προκαλεί περιττές αφυπνίσεις και αλλαγές πλαισίου εκτέλεσης, με αποτέλεσμα μειωμένη απόδοση.

4. Το παράλληλο πρόγραμμα δεν εμφανίζει σημαντική επιτάχυνση. Αυτό οφείλεται στο γεγονός ότι πολλά νήματα αναλαμβάνουν να υπολογίσουν και να εξάγουν διάφορες γραμμές του συνόλου. Ωστόσο, ο μηχανισμός συγχρονισμού που χρησιμοποιείται με τη χρήση των αμοιβαίων αποκλεισμών εισάγει περιττή επιβάρυνση και ανταγωνισμό. Κάθε νήμα αναμένει προτού προχωρήσει να εξάγει την υπολογισμένη γραμμή του. Το πρώτο νήμα επιτρέπεται να ξεκινήσει αμέσως με την αρχικοποίηση του πρώτου αμοιβαίου αποκλεισμού σε τιμή 1, ενώ οι υπόλοιποι αμοιβαίοι αποκλεισμοί αρχικοποιούνται σε τιμή 0.

5. Πατώντας Ctrl-C κατά τη διάρκεια της εκτέλεσης του προγράμματος, τότε δεν έχουν γίνει reset τα χρώματα και το τερματικό εμφανίζει ότι γράφουμε με χρώμα(το χρώμα στο οπίο διακόπηκε). Αυτό διορθώνεται εισάγωντας την εντολή reset_xterm_color() μέσα στο sigint handler.