ΚΩΤΣΗΣ ΣΤΑΘΗΣ 03115408
ΡΑΓΚΟΥΣΗΣ ΗΛΙΑΣ 03117897

Λειτουργικά Συστήματα
6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 3:
Συγχρονισμός

1.1
Συγχρονισμός σε υπάρχοντα κώδικα

```
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ make simplesync-
simplesync-atomic    simplesync-mutex
simplesync-atomic.o  simplesync-mutex.o
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ make simplesync-
```

Παρατηρούμε ότι μπορούν να παραχθούν 2 εκτελέσιμα το simplesync-atomic και simplesync-mutex από το ίδιο αρχείο πηγαίου κώδικα.
Αυτό συμβαίνει λόγω του dynamic library DSYNC -ATOMIC ή DSYNC-MUTEX που φορτώνουμε στον gcc για compilation μέσω του Makefile.

```
1  ## Simple sync (two versions)
2  simplesync-mutex: simplesync-mutex.o
3    $(CC) $(CFLAGS) -o simplesync-mutex simplesync-mutex.o $(LIBS)
4
5  simplesync-atomic: simplesync-atomic.o
6    $(CC) $(CFLAGS) -o simplesync-atomic simplesync-atomic.o $(LIBS)
7
8  simplesync-mutex.o: simplesync.c
9    $(CC) $(CFLAGS) -DSYNC_MUTEX -c -o simplesync-mutex.o simplesync.c
0
1  simplesync-atomic.o: simplesync.c
2    $(CC) $(CFLAGS) -DSYNC_ATOMIC -c -o simplesync-atomic.o simplesync.c
```

ΕΡΩΤΗΣΕΙΣ :

Έτσι παράγονται 2 εκτελέσιμα:Συγχρονισμός με mutex

```
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
```

και με atomic operations

```
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
```

1)Στην συνέχεια τρέχουμε το time ./simplesync-atomic , time ./simplesync-mutex πριν

```
oslaba25@os-node1:~/ask3/sync$ time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -7788865.

real    0m0.038s
user    0m0.072s
sys     0m0.000s
oslaba25@os-node1:~/ask3/sync$ time ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 7389620.

real    0m0.038s
user    0m0.072s
sys     0m0.000s
oslaba25@os-node1:~/ask3/sync$
```

Παρατηρούμε ότι πριν τον συγχρονισμό και ενώ βγάζουμε λάθος αποτελέσματα οι simplesync-mutex και simplesymc-atomic βγάζουν ίδια αποτελέσματα χρόνου ,πιο γρήγορα από τα αντιστοιχα με συγχρονισμό ,ενώ μετά τον επιτυχή συγχρονισμό το simplesync-mutex είναι πιο αργό από το simplesync-atomic .Αυτό είναι λογικό εφόσον δεν χάνουμε χρόνο για τον συγχρονισμό των threads τα οποία απλώς κάνουν λάθος read write και όχι load και δεν ξοδεύεται χρόνος για ορισμό-ανάθεση μνήμης στα mutex ή για τα atomic operations.

2)

2)Οι ατομικές λειτουργίες αξιοποιούν την υποστήριξη του επεξεργαστή (σύγκριση και ανταλλαγή οδηγιών) και δεν χρησιμοποιούν κλειδώματα, το locking/unlocking των οποίων μπορεί να δοηγήσει σε oveheard κατά το context switch.Αντίθετα, τα νήματα που επιχειρούν ατομικές λειτουργίες δεν περιμένουν και συνεχίζουν να προσπαθούν μέχρι την επιτυχία (το λεγόμενο busy-waiting), οπότε

αποφεύγεται το overhead του content-switching.Γι αυτό παρατηρούμε μεγαλύτερη ταχύτητα με atomic operations.

```
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.

real    0m0,091s
user    0m0,179s
sys     0m0,000s
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ 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    0m1,460s
user    0m1,928s
sys     0m0,916s
```

3)Τρέχοντας τις εντολές για την παραγωγή της Assembly έχουμε :

```
st:~/Downloads/os_lab_2020/ERG_ASK3/ask1.1$ gcc -DSYNC_MUTEX -S -c simplesync.c
```

```
$ gcc -DSYNC_ATOMIC -O2 -S -c simplesync.c
```

τα εξής :

```
decrease_fn:
.LFB54:
        .cfi_startproc
        subq    $24, %rsp
        .cfi_def_cfa_offset 32
        leaq    .LC2(%rip), %rdx
        movl    $10000000, %ecx
        movq    %rdi, (%rsp)
        movq    stderr(%rip), %rdi
        movl    $1, %esi
        movq    %fs:40, %rax
        movq    %rax, 8(%rsp)
        xorl    %eax, %eax
        call    __fprintf_chk@PLT
        movl    $10000000, %eax
        .p2align 4,,10
        .p2align 3
```

```
increase_fn:
.LFB53:
        .cfi_startproc
        subq    $24, %rsp
        .cfi_def_cfa_offset 32
        leaq    .LC0(%rip), %rdx
        movl    $10000000, %ecx
        movq    %rdi, (%rsp)
        movq    stderr(%rip), %rdi
        movl    $1, %esi
        movq    %fs:40, %rax
        movq    %rax, 8(%rsp)
        xorl    %eax, %eax
        call    __fprintf_chk@PLT
        movl    $10000000, %eax
        .p2align 4,,10
        .p2align 3
```

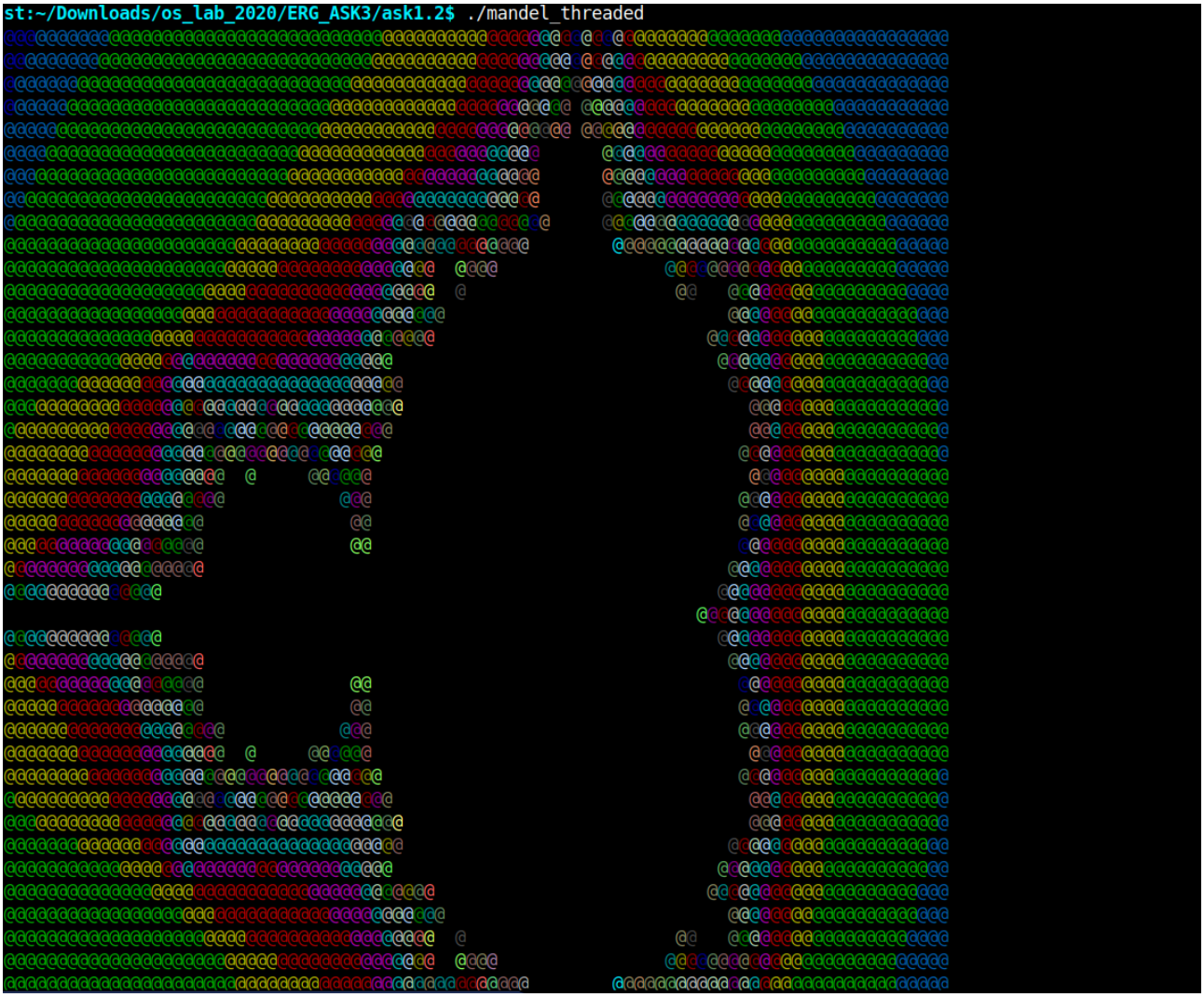
Για τα atomic operations
έχουμε τα παραπάνω (φορτώνει την τρέχουσα τιμή και αυξάνει/μειώνει κατά 1

4)ενώ για τα mutex :

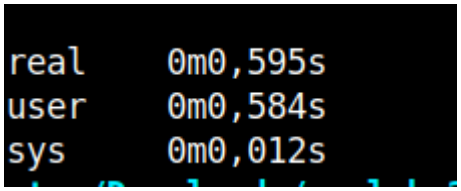
movl $mutex1, %edi
call pthread_mutex_lock
για την εντολή ->pthread_mutex_lock(&mutex1);
ενώ έχουμε

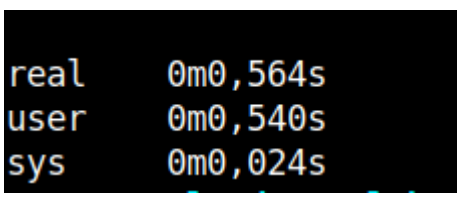
movl $mutex1, %edi
call pthread_mutex_unlock
για την εντολή →pthread_mutex_unlock(&mutex1);
(φορτώνει και κλειδώνει/ξεκλειδώνει τους καταχωρητές).

1.2
Παράλληλος υπολογισμός του συνόλου Mandelbrot



Ερωτήσεις:
1)Χρειαζόμαστε N σημαφόρους όπου N ο αριθμός των ζητούμενων threads στο command line εφόσον κάθε thread κάνει wait στον σημαφόρο του και μόλις τυπώσει μια γραμμή κάνει post στον σημαφόρο του thread της επόμενης γραμμής (next step).
2) Σειριακά Με 2 threads (με βελτίωση του critical section) (διπύρηνο pc)





3)Αρχικά, το παράλληλο πρόγραμμα ήταν πιο αργό. Αυτό οφειλόταν στο γεγονός ότι κάθε thread αναμένει για να υπολογίσει και εκτυπώσει την κάθε γραμμή.Αν όμως κάθε thread υπολογίσει τη γραμμή χωρίς να περιμένει και απλώς εκτυπώσει μόνο όταν έρθει η σειρά του, τότε το πρόγραμμα εμφανίζει επιτάχυνση. Το critical section πρέπει να έχει λοιπόν μόνο την εκτύπωση του τελικού υπολογισμού και όχι τον υπολογισμό.

4)Η reset_xterm_color() είναι η εντολή που επαναφέρει το χρώμα του terminal. Αν λοιπόν κάνουμε ctrl-c στο πρόγραμμα πριν ολοκληρωθεί δεν θα φτάσει εκεί και δεν θα κληθεί αυτή η συνάρτηση.Με την εισαγωγή ενός sigint_handler (SIGINT) που θα καλεί την reset_xterm_color() κάθε φορά που κάνουμε ctrl-c μπορούμε να διορθώσουμε το πρόβλημα αυτό.

ΚΩΔΙΚΕΣ

ΑΣΚ1.1

```
/*
 * 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
```

```
/* INITIALIZE THE MUTEX LOCK */
pthread_mutex_t mutex1=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++) {
		if (USE_ATOMIC_OPS) {
			/* ... */
			/* You can modify the following line */
			//	++(*ip);
			//atomic operation to add
			__sync_fetch_and_add(&ip,1);
			/* ... */
		} else {

			/*try to lock the mutex for incrementing*/
			pthread_mutex_lock(&mutex1);


			/* You cannot modify the following line */
			++(*ip);

			/* try to unlock the mutex after incrementing */
			pthread_mutex_unlock(&mutex1);

		}
	}
	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 */
                //              --(*ip);
                //atomic operation to subtract
                __sync_fetch_and_sub(&ip,1);
                /* ... */
            } else {
                /* try to lock mutex for decrementing  */
                pthread_mutex_lock(&mutex1);

                /* You cannot modify the following line */
                --(*ip);

                /* try to unlock mutex after decrementing */
                pthread_mutex_unlock(&mutex1);
            }
        }
        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;
}
```

ΑΣΚ1.2

```
/*
 * 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 <signal.h>
#include <semaphore.h>

#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
*/
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;


/*struct for thread info */
struct thread_info_struct {
pthread_t tid; /* POSIX thread id, as returned by the library */
sem_t *sem;
int thrid; /* Application-defined thread id */
int thrcnt;
};

/* function for dynamic memory allocation */
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;
}



/*atoi function */
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;
}
```

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

	compute_mandel_line(line, color_val);
	output_mandel_line(fd, color_val);
}
```

```
/*function that each thread implements */

void *thread_function (void* arg){
		int i ,current, next, color_val[x_chars];
		struct thread_info_struct *thr = arg;   //creating pointer to the info struct for each
thread
		current=thr->thrid;
		next=(current+1)%thr->thrcnt;  //the nexr thread


	for(i=thr->thrid;i<y_chars;i+= thr->thrcnt)		{
		compute_mandel_line(i, color_val);
		sem_wait(&thr->sem[current]);
		output_mandel_line(1, color_val);

		if (next < thr->thrcnt) {
				sem_post(&thr->sem[next]);
		}


	}
}
```

```
void sigint_handler() {
reset_xterm_color(1);
exit(1);
```

```
}


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


        int thrcnt,i,ret,temp;
        struct thread_info_struct *thr;  //the struct info
        //int nthreads=argv[1];


        sem_t * sem ;
        struct sigaction sa;

        sa.sa_handler = sigint_handler;
        sigaction(SIGINT, &sa, NULL);

        if(argc!=2){

                perror ("you need to give one parameter : the number of threads!");
                exit(1);
        }




        thr = safe_malloc(thrcnt * sizeof(*thr));
        sem = safe_malloc(thrcnt * sizeof(*sem));


        int line;

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


        /*semaphore initialization */
        for (i = 0; i < thrcnt; i++) {
                        if (i == 0) {
                                temp = 1;
                        }
                        else {
                                temp = 0;
                        }
                        if (sem_init(&sem[i], 0, temp) == -1) {
                                fprintf(stderr, "Error creating semaphore.\n");
                        }
 }
```

```
  /* creating the threads */
        for (i = 0; i < thrcnt; i++) {
                        thr[i].thrid = i;
                        thr[i].thrcnt = thrcnt;
                        thr[i].sem = sem;
                        /* Spawn new thread */
                        ret = pthread_create(&thr[i].tid, NULL, thread_function, &thr[i]);
                        if (ret) {
                                perror_pthread(ret, "pthread_create");
                                exit(1);}
                        }

                        /*wait for threads to end */

                        for (i = 0; i < thrcnt; i++) {
                                ret = pthread_join(thr[i].tid, NULL);
                                if (ret) {
                                        perror_pthread(ret, "pthread_join");
                                        exit(1);
                                }
                         }




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