Λειτουργικά Συστήματα – Άσκηση 3

Κυριακόπουλος Γιώργος – el18153
Τζελέπης Σεραφείμ – el18849

1.1

Πηγαίος κώδικας:

```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
```

```c
#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 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) {
            /* ... */
            /* You can modify the following line */
            __sync_add_and_fetch(ip, 1);
            /* ... */
        } else {
            /* ... */
            ret = pthread_mutex_lock(&mutex);
            if (ret) {
                perror_pthread(ret, "pthread_mutex_lock");
                exit(1);
            }
            /* You cannot modify the following line */
            ++(*ip);
            /* ... */
            ret = pthread_mutex_unlock(&mutex);
            if (ret) {
                perror_pthread(ret, "pthread_mutex_unlock");
                exit(1);
            }
        }
    }
    fprintf(stderr, "Done increasing variable.\n");

    return NULL;
}
```

```c
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_sub_and_fetch(ip, 1);
            /* ... */
        } else {
            /* ... */
            ret = pthread_mutex_lock(&mutex);
            if (ret) {
                perror_pthread(ret, "pthread_mutex_lock");
                exit(1);
            }
            /* You cannot modify the following line */
            --(*ip);
            /* ... */
            ret = pthread_mutex_unlock(&mutex);
            if (ret) {
                perror_pthread(ret, "pthread_mutex_unlock");
                exit(1);
            }
        }
    }
    fprintf(stderr, "Done decreasing variable.\n");

    return NULL;
}


int main(int argc, char *argv[])
{
    int val, ret, ok;
    pthread_t t1, t2;

    /*
     * Initial value
     */
```

```c
    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;
}
```

Έξοδος εκτέλεσης προγράμματος:

```
oslaba40@os-node1:~/ex3$ ./simplesync-atomic
        About to increase variable 10000000 times
        About to decrease variable 10000000 times
        Done increasing variable.
        Done decreasing variable.
        OK, val = 0.

oslaba40@os-node1:~/ex3$ ./simplesync-mutex
        About to increase variable 10000000 times
        About to decrease variable 10000000 times
        Done increasing variable.
        Done decreasing variable.
        OK, val = 0.
```

Ερωτήσεις:

Αρχικά, παρατηρούμε ότι αφού μεταγλωττίσουμε το πρόγραμμα, η έξοδος των δύο εκτελέσιμων είναι NOT OK, ακολουθούμενη από μία εμφανώς τυχαία τιμή της μεταβλητής val του προγράμματος. Αυτό συμβαίνει διότι στον κώδικα μας έχουμε δύο νήματα τα οποία εκτελούν κάποια αριθμητική πράξη στην ίδια μεταβλητή, προκαλώντας έτσι ένα race condition. Ως αποτέλεσμα, λόγω του τυχαίου τρόπου επιλογής σειράς εκτέλεσης, η τιμή της μεταβλητής val, παίρνει κάθε φορά τυχαίες τιμές και όχι την τιμή 0 που θα περιμέναμε μετά από ισόποσες αυξήσεις και μειώσεις της.

Όπως βλέπουμε με την χρήση της εντολής make προκύπτουν δυο εκτελέσιμα αρχεία από το ίδιο αρχείο κώδικα simplesync.c. Το γεγονός αυτό οφείλεται στην χρήση του -D flag και ενός μέρους του κώδικας μας, ο οποίος επιτρέπει μέσω του ορισμού ενός macro (-D[name]) να διαλέξουμε για κάθε αρχείο τους μηχανισμούς εκτέλεσης του κώδικα μας, με βάση τα if conditions και το flag USE_ATOMIC_OPS. Αυτό λαμβάνει τιμή 1 εάν έχουμε ορίσει ως macro το SYNC_ATOMIC και τιμή 0 εάν έχουμε ορίσει ως macro το SYNC_MUTEX, μέσω της εντολής -DSYNC_{ATOMIC,MUTEX} μέσα στο Makefile μας.

1. Με συγχρονισμό:

```
oslaba40@os-node1:~/ex3$ time ./simplesync-atomic
        About to increase variable 10000000 times
        About to decrease variable 10000000 times
        Done decreasing variable.
        Done increasing variable.
        OK, val = 0.
```

```
real    0m0.421s
user    0m0.832s
sys     0m0.004s

oslaba40@os-node1:~/ex3$ 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    0m26.772s
        user    0m27.092s
        sys     0m26.440s
```

Χωρίς συγχρονισμό:

```
oslaba40@os-node1:~/ex3$ time ./simplesync-atomic
        About to increase variable 10000000 times
        About to decrease variable 10000000 times
        Done increasing variable.
        Done decreasing variable.
        NOT OK, val = -3374558.

        real    0m0.039s
        user    0m0.072s
        sys     0m0.000s

oslaba40@os-node1:~/ex3$ time ./simplesync-mutex
        About to increase variable 10000000 times
        About to decrease variable 10000000 times
        Done increasing variable.
        Done decreasing variable.
        NOT OK, val = -9991389.

        real    0m0.039s
        user    0m0.072s
        sys     0m0.000s
```

Παρατηρούμε πως ο χρόνος που απαιτούν τα εκτελέσιμα με συγχρονισμό είναι μεγαλύτερος από αυτόν που απαιτούν τα εκτελέσιμα χωρίς συγχρονισμό, καθώς στην υλοποίηση με συγχρονισμό, υπάρχει η καθυστέρηση που προκαλείται είτε από την ενεργό αναμονή είτε από

τις κλήσεις συστήματος, αφού έχουμε μόνο ένα νήμα κάθε φορά σε κάθε κρίσιμο τμήμα του κώδικα, σε αντίθεση με την περίπτωση που δεν έχουμε συγχρονισμό και όλα τα νήματα μπορούν να έχουν πρόσβαση στο κρίσιμο τμήμα ταυτόχρονα.

2. Παρατηρούμε ότι η χρήση ατομικών λειτουργιών είναι πιο γρήγορη από την χρήση POSIX mutexes, διότι έχουμε μόνο δύο νήματα που διεκδικούν την είσοδο στο κρίσιμο τμήμα κώδικα όπου εκτελείται μόνο μία αύξηση ή μείωση μιας μεταβλητής, επομένως νήματα με πολύ μικρά διαστήματα αναμονής. Σε αυτήν την περίπτωση, συμφέρει η χρήση των ατομικών λειτουργιών και όχι η χρήση του αμοιβαίου αποκλεισμού με αναστολή εκτέλεσης, η οποία χρησιμοποιεί πολλαπλές κλήσεις συστήματος, σε αντίθεση με την πρώτη που χρησιμοποιεί μόνο απλές εντολές Assembly χωρίς να αναστέλλει και να ξυπνάει τα νήματα.

3. Για να παράγουμε το αρχείο Assembly χρησιμοποιούμε την εντολή:

```
oslaba40@os-node1:~/ex3$ gcc -Wall -O2 -pthread -DSYNC_ATOMIC
-S -g -o simplesync-atomic.s simplesync.c
```

Για την ατομική λειτουργία __sync_add_and_fetch(ip, 1):

```
.loc 1 51 4 view .LVU17
lock addl       $1, (%rbx)
```

Για την ατομική λειτουργία __sync_sub_and_fetch(ip, 1):

```
.loc 1 85 4 view .LVU47
lock subl       $1, (%rbx)
```

4. Για να παράγουμε το αρχείο Assembly χρησιμοποιούμε την εντολή:

```
oslaba40@os-node1:~/ex3$ gcc -Wall -O2 -pthread
-DSYNC_MUTEX -S -g -o simplesync-mutex.s simplesync.c
```

Για τα δύο σημεία που καλούμε την pthread_mutex_lock():

```
.loc 1 55 10 is_stmt 0 view .LVU22
movq   %r13, %rdi
call    pthread_mutex_lock@PLT

.loc 1 89 10 is_stmt 0 view .LVU81
movq   %r13, %rdi
call    pthread_mutex_lock@PLT
```

1.2

Πηγαίος κώδικας:

```
/*
 * 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 <pthread.h>
#include <semaphore.h>
#include <signal.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, ym
in)
*/
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;

int thrcnt = 3;
sem_t *mutex;

/*
 * 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 */
    int *color_val; /* Pointer to array to manipulate, used to hold
                     * color values for the line being drawn */
    int thrid; /* Application-defined thread id */
};

void intHandler(int dummy) {
    reset_xterm_color(1);
    printf("\n"); /* Print new line to seperate "^C" output" */
    exit(130); /* Exit status code 130, termination by Ctlr-C" */
}

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 \n\n"
        "Exactly one argument required:\n"
        "    thread_count: The number of threads to create.\n",
        argv0);
    exit(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(void *arg)
{
    int i;
    /* We know arg points to an instance of thread_info_struct */
    struct thread_info_struct *thr = arg;

    for (i = thr->thrid; i < y_chars; i += thrcnt){
        int current = i % thrcnt;
        int next = (i + 1) % thrcnt;
```

```
        compute_mandel_line(i, thr->color_val);
        sem_wait(&mutex[current]);

        output_mandel_line(1, thr->color_val);
        sem_post(&mutex[next]);
    }
    return NULL;
}

int main(int argc, char **argv)
{
    signal(SIGINT, intHandler);

    int i, ret;
    struct thread_info_struct *thr;

    if (argc != 2) {
        usage(argv[0]);
    }
    if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0) {
        fprintf(stderr, "`%s' is not valid for `thread_count'\n", arg
v[1]);
        exit(1);
    }

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

    thr = safe_malloc(thrcnt * sizeof(*thr));
    mutex = safe_malloc(thrcnt * sizeof(sem_t));

    for(i = 0; i < thrcnt; i++) {
        /* Initialize per-thread structure */
        thr[i].thrid = i;
        thr[i].color_val = safe_malloc(x_chars * sizeof(int));

        if(i == 0) {
            sem_init(&mutex[i], 0, 1);
        }
        else{
            sem_init(&mutex[i], 0, 0);
        }
```

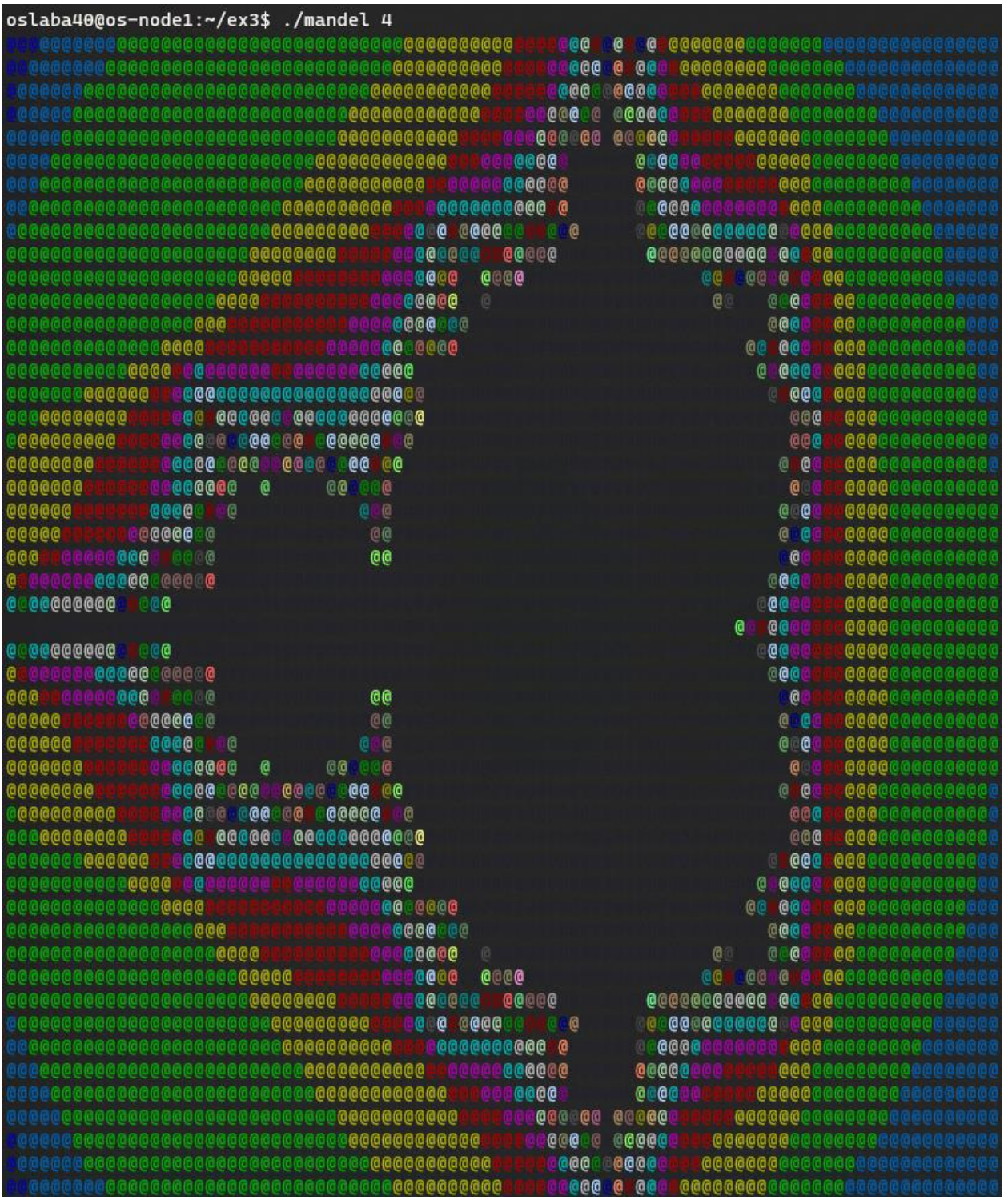
```
        /* Spawn new thread */
        ret = pthread_create(&thr[i].tid, NULL, compute_and_output_ma
ndel_line, &thr[i]);
        if (ret) {
            //perror_pthread(ret, "pthread_create");
            exit(1);
        }
    }

    /*
     * 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);
        }
    }

    for(i = 0; i < thrcnt; i++) {
        sem_destroy(&mutex[i]);
    }

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

Έξοδος εκτέλεσης προγράμματος:



Ερωτήσεις:

1. Χρειαζόμαστε πλήθος σημαφόρων ίσο με τον αριθμό των νημάτων (thrcnt), ο οποίος δίνεται μέσω του command line από τον χρήστη. Χρησιμοποιούμε ένα σημαφόρο για κάθε νήμα και θέλουμε το i-οστό νήμα να αναλαμβάνει τον υπολογισμό και την εκτύπωση της i, i+thrcnt, i+2*thrcnt-οστής κτλπ γραμμής. Επομένως κάθε σημαφόρος αφού το αντίστοιχο του νήμα υπολογίσει και εκτυπώσει (έχοντας πάρει άδεια από το προηγούμενο νήμα ή εάν πρόκειται για το πρώτο νήμα), δίνει άδεια στο επόμενο του νήμα να εκτυπώσει (αφού προηγουμένως έχει υπολογίσει τη γραμμή), ώστε να διατηρηθεί η σωστή σειρά εκτύπωσης και στη συνέχεια αναλαμβάνει τον υπολογισμό της i+x*thrcnt-οστής γραμμής και αναμένει μέχρι να πάρει άδεια εκτύπωσης από το προηγούμενο νήμα.

2. Οι παρακάτω μετρήσεις έγιναν σε σύστημα με 2 πυρήνες:

Παράλληλος υπολογισμός με 2 νήματα:

$time ./mandel 2

real 0m0.666s
user 0m1.022s
sys 0m0.221s

Σειριακός υπολογισμός:

$time ./mandel

real 0m1.190s
user 0m0.987s
sys 0m0.200s

3. Παρατηρούμε πως ο πραγματικός χρόνος εκτέλεσης είναι περίπου ο μισός στην περίπτωση παράλληλου υπολογισμού με 2 νήματα, ενώ οι χρόνοι χρήσης επεξεργαστή από το σύστημα και από το χρήστη παραμένουν σχεδόν ίσοι.

Στο πρόγραμμα μας το κρίσιμο τμήμα αποτελεί ουσιαστικά η εκτύπωση της κάθε γραμμής. Ο υπολογισμός γίνεται από τη στιγμή που το εκάστοτε νήμα εκτυπώσει την προηγούμενη του γραμμή που του αντιστοιχεί και όχι όταν πάρει το νήμα αυτό την άδεια, μέσω του sem_post από το νήμα της προηγούμενης γραμμής, κάτι που θα καθυστερούσε περισσότερο το πρόγραμμα μας και θα το μετέτρεπε ουσιαστικά σε μία παραλλαγή του σειριακού υπολογισμού.

4. Εάν πατήσουμε Ctrl-C την ώρα που εκτελείται το πρόγραμμα, σταματάει η εκτύπωση του Mandelbrot και δεν γίνεται reset στο χρώμα της γραμματοσειράς του τερματικού μας και επομένως, το τερματικό μας μετά την έξοδο μένει με το χρώμα του τελευταίου χαρακτήρα που είχε εκτυπωθεί κατά την εκτέλεση του προγράμματος.

Για να το αποφύγουμε αυτό, χρησιμοποιούμε έναν signal handler για να πιάσουμε το Ctrl-C μέσω του αντίστοιχου σήματος SIGINT, ο οποίος επαναφέρει το χρώμα του τερματικού, εκτυπώνει μια νέα σειρά για να χωρίσει την έξοδο της διακοπής "^C" από την επόμενη γραμμή του τερματικού και τερματίζει με τον κωδικό εξόδου 130, ο οποίος αντιστοιχεί σε διακοπή του χρήστη μέσω Ctrl-C.