Λειτουργικά Συστήματα

Άσκηση 3: Συγχρονισμός

Ομάδα: oslabd43

Ο/Ε: Αντώνης Παπαοικονόμου 03115140

Γιάννης Πιτόσκας 03115077

1.1 Συγχρονισμός σε υπάρχοντα κώδικα

Κώδικας `simple-sync.c`:

```
/*
 * simplesync.c
 *
 * A simple synchronization exercise.
 *
 * Antonis Papaoikonomou
 * Giannis Pitoskas
 * 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 mutex = 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 */
                        __sync_add_and_fetch(ip,1);
                        /* ... */
                } else {
                        pthread_mutex_lock(&mutex);
                        /* ... */
                        /* You cannot modify the following line */
                        ++(*ip);
                        /* ... */
                        pthread_mutex_unlock(&mutex);
                }
        }
        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 */
                        __sync_sub_and_fetch(ip,1);
                        /* ... */
                } else {
                        pthread_mutex_lock(&mutex);
                        /* ... */
                        /* You cannot modify the following line */
                        --(*ip);
                        /* ... */
                        pthread_mutex_unlock(&mutex);
                }
        }
        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;
}
```

Ερωτήσεις:

1. **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;**

Χρησιμοποιώντας την εντολή `time` λαμβάνουμε το εξής αποτελέσματα για τον χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό:

```
oslabd43@os-node2:~/Ask3$ time ./simplesync-nosync
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 3456635.

real    0m0.094s
user    0m0.064s
sys     0m0.000s
```

(No sync)

Χρησιμοποιώντας την `time` και στα δύο εκτελέσιμα που εκτελούν συγχρονισμό λαμβάνουμε τα εξής αποτελέσματα για τον χρόνο εκτέλεσης του καθενός εκ των δύο:

```
oslabd43@os-node2:~/Ask3$ 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    0m1.337s
user    0m1.032s
sys     0m0.080s
```

(Mutexes)

```
oslabd43@os-node2:~/Ask3$ 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    0m0.372s
user    0m0.332s
sys     0m0.000s
```

(Atomics)

Το αρχείο χωρίς συγχρονισμό παρουσιάζει τους μικρότερους χρόνους εκτέλεσης. Τους εν συνεχεία μικρότερους χρόνους εκτέλεσης τους παρουσιάζει το αρχείο που χρησιμοποιεί atomic operations και τέλος οι μεγαλύτεροι χρόνοι εκτέλεσης εμφανίζονται για το αρχείο που χρησιμοποιεί τα mutexes.

Οι παραπάνω διαφορές στους χρόνους εκτέλεσης μεταξύ “μη-συγχρονισμένου” και “συγχρονισμένων” οφείλονται στο ότι στο εκτελέσιμο που δεν χρησιμοποιεί συγχρονισμό δεν υπάρχει κάποια χρονική καθυστέρηση κατά την απουσία συγχρονισμού στην εκτέλεση των νημάτων (δεν ασκείται κάποιος περιορισμός) και γι’ αυτό είναι το ταχύτερο.

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

Το εκτελέσιμο που κάνει χρήση atomic operations είναι ταχύτερο από αυτό με τα mutexes, καθώς είναι πιο low level αφού μεταφράζει τις εντολές που πρέπει να εκτελεστούν σε μία μόνο εντολή assembly. Τέλος, το εκτελέσιμο που κάνει χρήση των mutexes εκτελούν συγχρονισμό σε πιο high level, αφού για τον συγχρονισμό γίνεται χρήση αλγορίθμου και έτσι παρουσιάζουν μεγαλύτερο χρόνο εκτέλεσης μιας και γίνεται μετάφραση των εντολών σε περισσότερες από μία εντολή assembly (τα mutexes χρησιμοποιούν atomic operations για την υλοποίηση τους).

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 .

Ενσωματώνουμε στο `Makefile` τον παρακάτω κανόνα `simplesync-atomic-asem` για την παραγωγή του ενδιάμεσου κώδικα assembly ο οποίος ύστερα από την εκτέλεση αυτού του κανόνα θα βρίσκεται στο αρχείο `simplesync-atomic.S`.

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

Τα κομμάτια κώδικα assembly που αντιστοιχούν στη μετάφραση της χρήσης ατομικών λειτουργειών φαίνεται παρακάτω:

```
.L2:
	.loc 1 51 0
	lock addl	$1, (%rbx)
```

```
.L7:
	.loc 1 77 0
	lock subl	$1, (%rbx)
```

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.**

Ενσωματώνουμε στο `Makefile` τον παρακάτω κανόνα `simplesync-mutex-asem` για την παραγωγή του ενδιάμεσου κώδικα assembly ο οποίος ύστερα από την εκτέλεση αυτού του κανόνα θα βρίσκεται στο αρχείο `simplesync-mutex.S`.

```
simplesync-mutex-asem: simplesync.c
	$(CC) $(CFLAGS)  -DSYNC_MUTEX simplesync.c -S -g -o simplesync-mutex.S
```

Ο αντίστοιχος κώδικας assembly που παράγεται για την `pthread_mutex_lock()` είναι ο ακόλουθος:

```
.L2:
	.loc 1 54 0
	movl	$mutex, %edi
	call	pthread_mutex_lock
.LVL4:
	.loc 1 57 0
	movl	0(%rbp), %eax
	.loc 1 59 0
	movl	$mutex, %edi
	.loc 1 57 0
	addl	$1, %eax
	movl	%eax, 0(%rbp)
	.loc 1 59 0
	call	pthread_mutex_unlock
```

1.2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

Κώδικας `mandel2.c`:

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
```

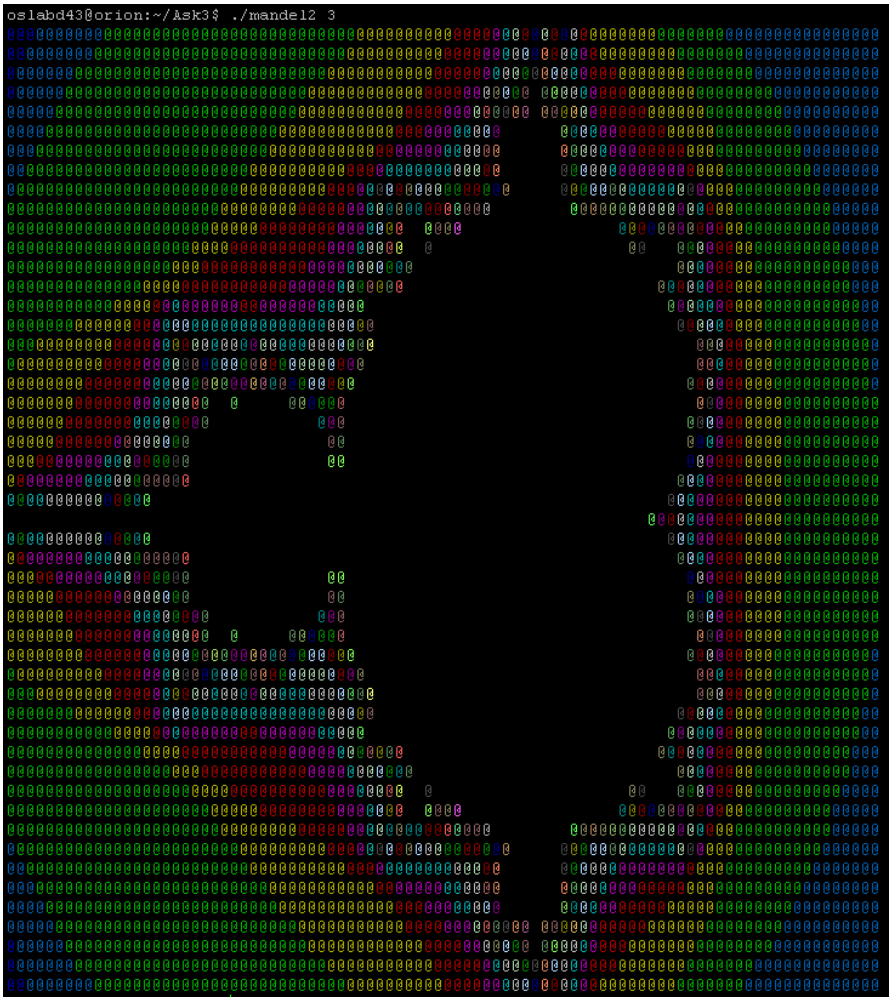
```
#include <math.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>
#include <semaphore.h>

#include "mandel-lib.h"

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

#define MANDEL_MAX_ITERATION 100000

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

struct thread_info_struct {
        pthread_t tid; /* POSIX thread id, as returned by the library */

        int thrid; /* Application-defined thread id */
        int N;
};

/*
 * 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.
 */

sem_t *sem;

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

void usage(char *argv0)
{
        fprintf(stderr, "Usage: %s NTHREADS \n\n"
                "Exactly one argument required:\n"
                "    NTHREADS: The number of threads to create.\n",
                argv0);
        exit(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;
}

int safe_papoi(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 *thread_compute(void *arg)
{
        struct thread_info_struct *thr = arg;
        int line;
        for (line = thr->thrid; line < y_chars; line+= thr->N) {
                compute_and_output_mandel_line(1, line, thr->N);
        }

        return NULL;
}

void Interrupt_Handler()
{
        reset_xterm_color(1);
        printf("\n");
        exit(1);
}

int main(int argc, char *argv[])
{
        int NTHREADS, ret;
        struct thread_info_struct *thr;

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

        thr = safe_malloc(NTHREADS * sizeof(*thr));
        sem = safe_malloc(NTHREADS * sizeof(*sem));
        int i;

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

        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.
         */

        // Incrementing the first semaphore
        sem_post(&sem[0]);

        for (i = 0; i < NTHREADS; i++) {
                thr[i].thrid = i;
                thr[i].N = NTHREADS;

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

        signal(SIGINT, Interrupt_Handler);       /* When USER -> CTRL-C */

        /*
         * Wait for all threads to terminate
         */

        for (i = 0; i < NTHREADS; i++) {
                ret = pthread_join(thr[i].tid, NULL);
                if (ret) {
                        perror_pthread(ret, "pthread_join");
```

```
                        exit(1);
                }
        }

        // Destroy semaphores
        for (i = 0; i < NTHREADS; i++){
                sem_destroy(&sem[i]);
        }

        free(sem);
        free(thr);

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

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

Έξοδος για $NTHREADS = 3$:



Ερωτήσεις:

1. ***Πόσοι σημαφόροι χρειάζονται για το σχήμα συγχρονισμού που υλοποιείτε;***

Στο σχήμα συγχρονισμού που έχουμε υλοποιήσει στο αρχείο `mandel2.c` χρειάζονται Ν σημαφόροι, όσοι και το πλήθος των threads που δημιουργούμε.

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.***

Τρέχοντας την εντολή `cat /proc/cpuinfo` στον `orion` παίρνουμε:

```
oslabd43@orion:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 2
model name      : QEMU Virtual CPU version 1.1.2
stepping        : 3
microcode       : 0x1000065
cpu MHz         : 2400.028
cache size      : 512 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 4
wp              : yes
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2
syscall nx lm nopl pni cx16 popcnt hypervisor lahf_lm svm abm sse4a vmmcall
bogomips        : 4800.05
TLB size        : 1024 4K pages
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 2
model name      : QEMU Virtual CPU version 1.1.2
stepping        : 3
microcode       : 0x1000065
cpu MHz         : 2400.028
cache size      : 512 KB
physical id     : 1
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 4
wp              : yes
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2
syscall nx lm nopl pni cx16 popcnt hypervisor lahf_lm svm abm sse4a vmmcall
bogomips        : 4800.05
TLB size        : 1024 4K pages
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

Για τον κώδικα με την σειριακή υλοποίηση παίρνουμε:

```
oslabd43@orion:~/Ask3$ time ./mandel
…
real    0m0.795s
user    0m0.748s
sys     0m0.016s
```

Ενώ για τον κώδικα με την παράλληλη υλοποίηση για 2 νήματα έχουμε:

```
oslabd43@orion:~/Ask3$ time ./mandel2 2
…
real    0m0.493s
user    0m0.748s
sys     0m0.028s
```

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;***

Παρατηρούμε ότι το πρόγραμμα με τον παράλληλο υπολογισμό του Mandelbrot με threads παρουσιάζει επιτάχυνση έναντι του σειριακού. Το κρίσιμο τμήμα του σχήματος συγχρονισμού που περιέχεται στο αρχείο `mandel2.c` περιέχει μόνο τη φάση του υπολογισμού και όχι αυτή της εξόδου κάθε γραμμής, καθώς στην δεύτερη απαιτείται σειριακή εκτέλεση για να είναι σωστό το αποτέλεσμα μας (κάτι το οποίο δεν απαιτείται στην φάση του υπολογισμού).

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;***

Όταν πατήσουμε τα πλήκτρα `Ctrl+C` στέλνεται στο πρόγραμμα μας ένα signal `SIGINT`. Παρατηρούμε πως αν πατήσουμε `Ctrl+C` κατά την διάρκεια εκτύπωσης του Mandelbrot μας τότε το τερματικό παραμένει στο χρώμα που είχε η τελευταία εκτύπωση πριν το signal. Για να το αποτρέψουμε αυτό θα πρέπει να τροποποιήσουμε τον signal handler έτσι ώστε όταν δέχεται το `SIGINT` να επαναφέρει το χρώμα στην default τιμή του και στη συνέχεια να τερματίζει. Αυτό μπορεί να γίνει με την εντολή `signal(SIGINT, Interrupt_Handler);`
Όπου η `Interrupt_Handler` φαίνεται παρακάτω:

```
void Interrupt_Handler()
{
        reset_xterm_color(1);
        printf("\n");
        exit(1);
}
```