

26 Μαΐου 2020



ΑΚΑΔΗΜΑΪΚΟ  
ΕΤΟΣ : 2019-  
2020

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Άσκηση 3η - Συγχρονισμός  
| Καπαρού Αλεξάνδρα (03117100)  
Χλαπάνης Οδυσσέας (03117023)  
Ομάδα : oslaba21

## Άσκηση 3.1

- Makefile

Μετά τη χρήση του Makefile παρατηρούμε ότι δημιουργούνται τρία εκτελέσιμα. Το pthreadtest.c το οποίο λειτουργεί κανονικά (τυπώνει ok) και τα simplesync-atomic και simplesync-mutex τα οποία δεν λειτουργούν κανονικά (not ok).

- Δύο εκτελέσιμα από ένα αρχείο κώδικα

Παρόλο που έχουμε μόνο δύο αρχεία κώδικα παράγονται τρία διαφορετικά εκτελέσιμα. Αυτό συμβαίνει επειδή στον κώδικα simplesync.c υπάρχει ένα if statement το οποίο εξαρτάται από δεδομένα τα οποία παρέχει το Makefile. Στην κατασκευή του Makefile επιλέγουμε να κάνουμε μεταγλώττιση του ενός προγράμματος με τον έναν τρόπο και του άλλου με τον άλλο (-DSYNC\_ATOMIC και -DSYNC\_MUTEX).

### ➤ Πηγαίος κώδικας της 3.1

```
/*
 * 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 */
/* ... */
pthread_mutex_t lock;
```

```

#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

```

```

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_fetch_and_add(ip,1);
            /* ... */
        } else {
            if(pthread_mutex_lock(&lock))
            {
                fprintf(stderr, "Mutex lock error");
                exit(1);
            }
            /* You cannot modify the following line */
            ++(*ip);
            //
            if(pthread_mutex_unlock(&lock))
            {
                fprintf(stderr, "Mutex unlock error");
            }
        }
    }
    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_fetch_and_sub(ip,1);
            /* ... */
        } else {
            if(pthread_mutex_lock(&lock))
        {
            fprintf(stderr, "Mutex lock error");
        }

            /* ... */
            --(*ip);
            /* ... */

            if(pthread_mutex_unlock(&lock))
            {
                fprintf(stderr, "Mutex unlock error");
            }

        }
    }
    fprintf(stderr, "Done decreasing variable.\n");

    return NULL;
}

```

```

int main(int argc, char *argv[])
{
    int val, ret, ok;
    pthread_t t1, t2;
    if(pthread_mutex_init( &lock,NULL))
    {
        perror("error in initialise of mutex");
        exit(1);
    }
    /*
    * 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);

    pthread_mutex_destroy(&lock);

    return ok;
}

```

## To Makefile που φτιάξαμε

CC = gcc

# CAUTION: Always use '-pthread' when compiling POSIX threads-based  
# applications, instead of linking with "-lpthread" directly.

CFLAGS = -Wall -O2 -pthread

LIBS =

all: simplesync-mutex simplesync-atomic

```

## 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

clean:
rm -f *.s *.o pthread-test simplesync-{atomic,mutex}

```

### ➤ Ερωτήσεις της 3.1

1. Βλέπουμε ότι λόγω των κλειδωμάτων κατά τον συγχρονισμό, δεν μπορούν να έχουν ταυτόχρονα πρόσβαση στο κρίσιμο τμήμα όλα τα threads και γι'αυτό τον λόγο υπάρχει παραπάνω καθυστέρηση σε σχέση με όταν δεν έχουμε συγχρονισμό.

```

oslaba21@os-nodel:~/3/3.ody/3.1$ 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    0m0.411s
user    0m0.808s
sys     0m0.000s
oslaba21@os-nodel:~/3/3.ody/3.1$ 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    0m3.640s
user    0m3.640s
sys     0m2.796s

```

```

oslaba21@os-nodel:~/3/3.alex/3.1$ time ./simplesync-wrong
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
NOT OK, val = -7157200.

real    0m0.038s
user    0m0.072s
sys     0m0.000s

```

2. Παρατηρούμε ότι το αρχικό λάθος πρόγραμμα είναι το πιο γρήγορο (real time) ενώ το mutex πρόγραμμα είναι το πιο αργό. Αυτό συμβαίνει μάλλον επειδή το λάθος πρόγραμμα εκτελεί όλα τα threads τελείως παράλληλα (με δυσάρεστα αποτελέσματα βέβαια) χωρίς να περιμένει κάτι, το atomic περιμένει κάθε thread αλλά πάντα κάποιο thread κάνει πρόοδο, ενώ το mutex δεν μας βεβαιώνει ότι κάποιο thread κάνει πρόοδο πάντα.
3. Ένα παράδειγμα της assembly που δημιουργείται είναι αυτό της ατομικής πρόσθεσης που φαίνεται στην παρακάτω φωτογραφία, το οποίο μας δείχνει ότι στην γραμμή 50 (.loc 1 50 0) έχουμε κλείδωμα (lock) και προσθήκη του άσσου στον καταχωρητή rbx( δηλαδή αύξηση κατά 1). Κοιτάζοντας τον κώδικα simplesync.c βλέπουμε ότι σε εκείνη την γραμμή αντιστοιχεί η συνάρτηση \_sync\_fetch\_and\_add(ip,1) που πράγματι κλειδώνει τον καταχωρητή και τον αλλάζει.

```

.L2:
        .loc 1 50 0
        lock addl    $1, (%rbx)

```

4. Ένα κομμάτι κώδικα σε c για τα posix mutexes είναι το παρακάτω:

```

if(pthread_mutex_lock(&lock))
{
    fprintf(stderr, "Mutex lock error");
    exit(1);
}
/* You cannot modify the following line */
++(*ip);
//
if(pthread_mutex_unlock(&lock))
{
    fprintf(stderr, "Mutex unlock error");
}

```

Το οποίο μεταφράζεται σε assembly ως εξής:

```
.L19:
    .loc 1 53 0
    movl    $lock, %edi
    call    pthread_mutex_lock
.LVL22:
    testl   %eax, %eax
    jne     .L26
    .loc 1 59 0
    movl    0(%rbp), %eax
    .loc 1 61 0
    movl    $lock, %edi
    .loc 1 59 0
    addl    $1, %eax
    movl    %eax, 0(%rbp)
    .loc 1 61 0
    call    pthread_mutex_unlock
```

Βλέπουμε δηλαδή ότι έχουμε τις συναρτήσεις `pthread_mutex_lock` και `pthread_mutex_unlock` που κλειδώνουν και ξεκλειδώνουν αντίστοιχα το κρίσιμο μέρος. Παρατηρούμε επίσης ότι πριν την αύξηση `++(*ip)` έχουμε δύο εντολές μετακίνησης `movl` που μεταφέρουν κάπου την τιμή (`movl 0(%rbp), %eax`) και την κλειδώνουν (`$lock, %edi`) και μετά την αύξηση έχουμε άλλη μια μετακίνηση που επαναφέρει την τιμή (`%eax, 0(%rbp)`). Καταλαβαίνουμε λοιπόν ότι γίνονται κάποιες άσκοπες μετακινήσεις κατά την υλοποίηση με `mutexes` κάτι το οποίο γίνεται εμφανές από την αύξηση του χρόνου αυτής της υλοποίησης σε σχέση με εκείνη που χρησιμοποιεί ατομικές λειτουργίες, όπως είδαμε και στο 1ο ερώτημα.

### Άσκηση 3.2

#### ➤ Πηγαίος κώδικας της 3.2

```
/*
 * 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 <pthread.h>
#include <semaphore.h>
#include "mandel-lib.h"
#include <errno.h>
#include <signal.h>
#define perror_pthread(ret, msg) \
    do { errno = ret; perror(msg); } while (0)
#define MANDEL_MAX_ITERATION 100000

/*****
 * Compile-time parameters *
 *****/
void signalhandler(int ctrl)    //What to do if the user presses Ctrl-C
{
    signal(ctrl, SIG_IGN);      //SIG_IGN is used to ignore the signal
    reset_xterm_color(1);      //Reset color
    exit(0);
}

/*
 * Output at the terminal is x_chars wide by y_chars long
 */
int y_chars = 40;
int x_chars = 130;
int thread_no; // = 3;    //the number of threads

typedef struct    //rename the struct
{
    pthread_t thread_id; //id for every thread that is created
    int line;            //the line for every thread
    sem_t semaphore;     //semaphore between line and line+1
}pthread_struct;

pthread_struct *thread;

/*
 * 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 *compute_and_output_mandel_line(void *arg) //arg points to an
instance of pthread_struct
{
    int i; //i is the line we have now
    int line2=*(int *)arg; //line2 is the first line of the thread(0,1,...,n-1)
    //some explanation about pointers: arg points to
pthread_struct,
    //with (int *)arg we are dereferencing arg to get the
address of the
    //instance and with *(int *)arg we get the real value of the
instance
    for (i=line2;i<y_chars;i=i+thread_no) //the next line of the thread is
given by i,i+n,i+2n etc
    {
        int color_val[x_chars];
        compute_mandel_line(i, color_val); //compute i line
        sem_wait(&thread[(i%thread_no)].semaphore); //synchronization
begins,we wait until semaphore of the specific thread comes
        output_mandel_line(1, color_val); //draw the line to the standart
output
        sem_post(&thread[((i%thread_no)+1)%thread_no].semaphore);
//increase the semaphore of the next thread
    }
}
/*

```

An example to explain the above semaphores:  
We have 2 threads(thread\_no=2) so line 0,2,4,.. is painted by thread 0 and  
line 1,3,5,..  
is painted by thread 1.

*If for example we want to compute line 5(i=5),then we have to bring semaphore for thread 1(to lock) and we do it with i%thread\_no=1.Then we print.Then we unlock and we have to bring the semaphore for the next thread(thread 0 again) and we do it with ((i%thread\_no)+1)%thread\_no=(1+1)%2=0.*

```
*/  
return 0;  
}
```

```
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_no\n\n"  
        "Exactly one argument required:\n"  
        "  thread_no: The number of threads to create.\n",  
        argv0);  
    exit(1);  
}
```

```
int main(int argc, char *argv[])  
{  
    signal(SIGINT, signalhandler);
```

```
    int line2,create,join;
```

```
    if (argc != 2)  
    {  
        usage(argv[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.
 */
int threads = atoi(argv[1]);
thread_no=threads;

if (thread_no < 1 || thread_no > y_chars-1)
{
printf("Usage: %s thread_no\n\n"
"Out of borders:\n"
"  thread_no: The number of threads to create.\n",argv[0]);
exit(1);
}

thread=(pthread_struct
*)safe_malloc(thread_no*sizeof(pthread_struct));
sem_init(&thread[0].semaphore,0,1); //We initialise
semaphore(semaphore 0) with value 1
if (thread_no>1) //If we have many threads then we have the same
number of semaphores
{
for (line2 =1; line2 < thread_no;line2++)
{
sem_init(&thread[line2].semaphore,0,0); //these semaphores have
value 0 because they wait
}
}

for (line2=0;line2<thread_no;line2++) //We create thread_no threads
and each one goes to the
{
//respective line to begin from there
thread[line2].line=line2; //So if we have 2 threads then the
1st(thread[0])
//begins at line=0 and the second
//(thread[1]) begins at line=1

create=pthread_create(&thread[line2].thread_id,NULL,compute_and_output
_mandel_line,&thread[line2].line);
if (create) {
perror_pthread(create, "pthread_create");
exit(1);
}
}

for (line2=0;line2<thread_no;line2++) //We join threads for the output
{

```

```

        join=pthread_join(thread[line2].thread_id,NULL);
        if (join){
            perror_pthread(join, "pthread_join");
        }
    }
    reset_xterm_color(1);
    return 0;
}

```

## To Makefile που φτιάξαμε:

*CC = gcc*

*CFLAGS = -Wall -O2 -pthread*

*all: mandel*

*mandel: mandel.o mandel-lib.o*

*\$(CC) \$(CFLAGS) -o mandel mandel.o mandel-lib.o*

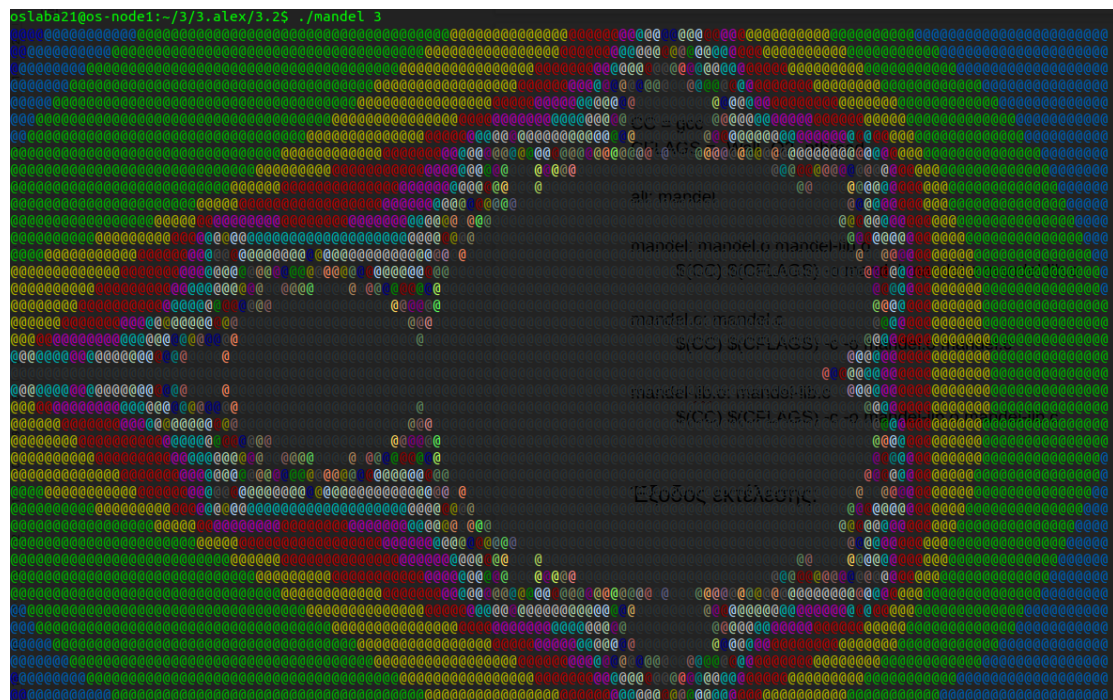
*mandel.o: mandel.c*

*\$(CC) \$(CFLAGS) -c -o mandel.o mandel.c*

*mandel-lib.o: mandel-lib.c*

*\$(CC) \$(CFLAGS) -c -o mandel-lib.o mandel-lib.c*

## ➤ Έξοδος εκτέλεσης προγραμμάτων της 3.2



```
oslaba21@os-node1:~/3/3.alex/3.2$ ./mandel
Usage: ./mandel thread_no

Exactly one argument required:
  thread_no: The number of threads to create.
```

```
oslaba21@os-node1:~/3/3.alex/3.2$ ./mandel 0
Usage: ./mandel thread_no

Out of borders:
  thread_no: The number of threads to create.
```

```
oslaba21@os-node1:~/3/3.alex/3.2$ ./mandel 40
Usage: ./mandel thread_no

Out of borders:
  thread_no: The number of threads to create.
```

### ➤ Ερωτήσεις της 3.2

1. Σε αυτή την άσκηση χρειαζόμαστε τόσους σημαφόρους όσους είναι και ο αριθμός των threads, καθώς κάθε thread έχει τον δικό του σημαφόρο προκειμένου να ξέρει πότε να τυπώσει αυτό και πότε να περιμένει τα υπόλοιπα threads να τυπώσουν.
2. Αρχικά γράφοντας την εντολή `cat /proc/cpuinfo` στο command line βλέπουμε ότι διαθέτουμε επεξεργαστή 4 πυρήνων:

```
oslaba21@os-node1:~/3/3.alex/3.2$ cat /proc/cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 23
model name     : Intel(R) Xeon(R) CPU           E5405  @ 2.00GHz
stepping       : 6
microcode      : 0x606
cpu MHz        : 2000.157
cache size     : 6144 KB
physical id    : 0
siblings       : 4
core id        : 0
cpu cores      : 4
apicid         : 0
initial apicid : 0
fpu            : yes
fpu_exception  : yes
cpuid level    : 10
wp             : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
                 pni dtes64 monitor ds_cpl vmx tm2 ssse3 cx16 xtpr pdcm dca sse4_1
bogomips       : 4000.31
clflush size   : 64
cache_alignment : 64
address sizes   : 38 bits physical, 48 bits virtual
power management:
```

Στην συνέχεια, χρησιμοποιώντας την εντολή time μπορέσαμε να χρονομετρήσουμε την εκτέλεση του σειριακού και του παράλληλου προγράμματος:

Για το σειριακό:

```
real    0m1.021s
user    0m0.968s
sys     0m0.028s
```

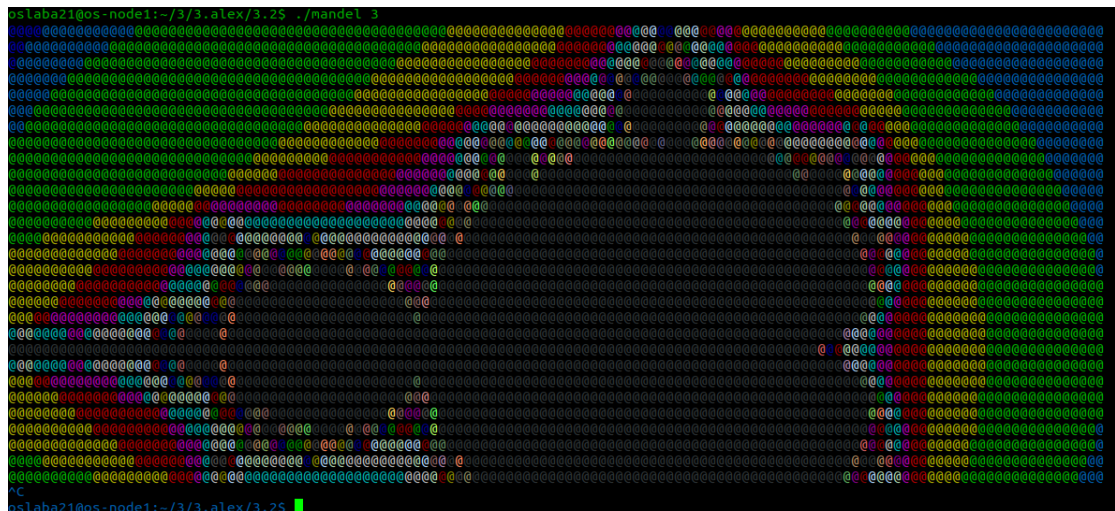
Για το παράλληλο με 2 threads:

```
real    0m0.610s
user    0m1.140s
sys     0m0.016s
```

Ενδεικτικά για το παράλληλο με 39 threads:

```
real    0m0.174s
user    0m1.136s
sys     0m0.024s
```

3. Λαμβάνοντας τα παραπάνω δεδομένα υπόψιν, βλέπουμε πως υπάρχει επιτάχυνση στο παράλληλο πρόγραμμα σε σχέση με το σειριακό. Πιο συγκεκριμένα, έχουμε χρησιμοποιήσει συγχρονισμό με την χρήση των `semaphores` προκειμένου να επιτύχουμε σωστή εκτύπωση ενώ ο υπολογισμός των γραμμών γίνεται παράλληλα.
4. Βλέπουμε ότι αν πατήσουμε Ctrl-C το terminal αλλάζει χρώμα και κρατάει το χρώμα που είχε υπολογιστεί για την τελευταία γραμμή, δηλαδή:





Γνωρίζοντας ότι για το Ctrl-C ισχύει:

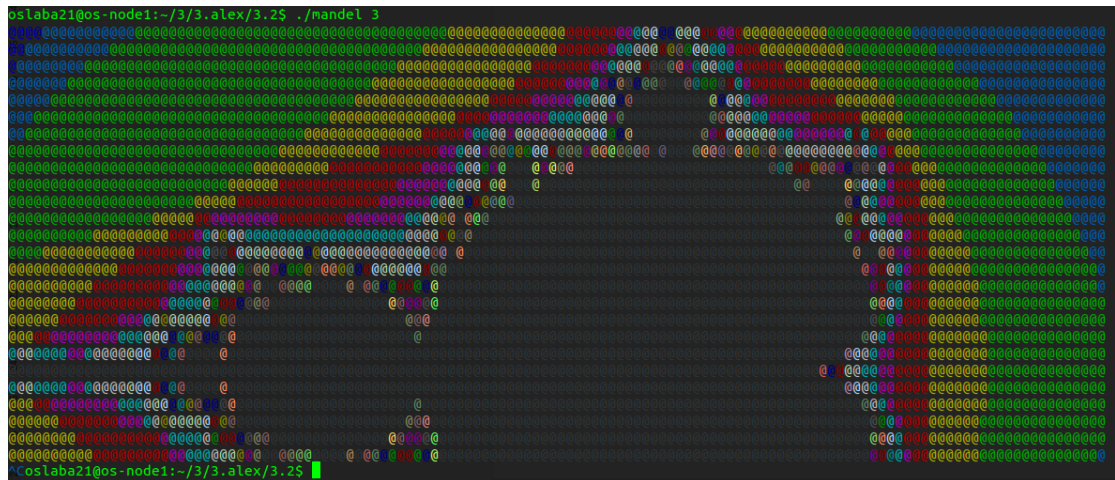
-   [is abort in UNIX](#):

In POSIX systems, the sequence causes the active program to receive a SIGINT signal. If the program does not specify how to handle this condition, it is terminated. Typically a program which does handle a SIGINT will still terminate itself, or at least terminate the task running inside it.

Θέλουμε να τροποποιήσουμε το πρόγραμμα μας ώστε μόλις λαμβάνει το σήμα SIGINT, να το αγνοούμε, να κάνουμε reset το χρώμα μέσω της δοθείσας συνάρτησης `reset_xterm_color(1)` και να τερματίζουμε το πρόγραμμα. Γι'αυτό φτιάχνουμε τον ακόλουθο signal handler:

```
void signalhandler(int ctrl)           //What to do if the user presses Ctrl-C
{
    signal(ctrl, SIG_IGN);             //SIG_IGN is used to ignore the signal
    reset_xterm_color(1);              //Reset color
    exit(-1);
}
```

Όπου πράγματι δουλεύει καθώς:



```
kslaba21@os-node1:~/3/3.alex/3.25$ ./mandel 3
kslaba21@os-node1:~/3/3.alex/3.25$
```