OSLab 2023-24 — 2η Εργαστηριακή Άσκηση

Κόρδας Νικόλαος - Α.Μ.: 03121032
Κριθαρίδης Κωνσταντίνος - Α.Μ.: 03121045

17 Μαΐου 2024

1 Συγχρονισμός σε υπάρχοντα κώδικα

Ζητούμενα

- Αρχικά χρησιμοποιούμε το δοσμένο Makefile για να μεταγλωττίσουμε τον κώδικα του αρχείου simplesync.c. Παρατηρούμε πως από αυτό το αρχείο προκύπτουν 2 εκτελέσιμα προγράμματα, πράγμα ασυνήθιστο, αφού συνήθως αναμένουμε ένα εκτελέσιμο από κάθε αρχείο. Επιπλέον, όταν εκτελούμε τόσο το simplesync_atomic όσο και το simplesync_mutex, η τιμή της μεταβλητής val είναι διάφορη του 0, αντιθέτως με ό,τι θα θέλαμε. Αυτό είναι αναμενόμενο, αφού τα διαφορετικά νήματα αλλάζουν την ίδια μεταβλητή val και προκύπτουν συνθήκες ανταγωνισμού στον συγχρονισμό του προγράμματος που οδηγούν σε τυχαία συμπεριφορά.

```
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ± ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 3645706.
```

```
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ± ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 77232.
```

- Αυτό συμβαίνει διότι κατά την δημιουργία των αρχείων simplesync-mutex.o και simplesync-atomic.o στο Makefile χρησιμοποιούνται κατά την μεταγλώττιση τα flags -DSYNC_MUTEX και -DSYNC_ATOMIC αντίστοιχα. Αυτό έχει ως συνέπεια, εσωτερικά στον κώδικα, όπου γίνεται έλεγχος για το ποιο από τις 2 flags χρησιμοποιήθηκε, (`#if defined (SYNC_ATOMIC) ^ defined(SYNC_MUTEX) == 0`), να έχουμε διαφορετικά κομμάτια κώδικα να μεταγλωττίσουμε. Το πρώτο κομμάτι αντιστοιχεί στη λύση του προβλήματος συγχρονισμού με atomic operations και το δεύτερο με mutex.

- Η επέκταση του κώδικα για τον συγχρονισμό των νημάτων τόσο με mutex όσο και με atomic operations φαίνεται στον κώδικα του αρχείου simplesync.c που θα ανεβάσουμε μαζί με αυτή την αναφορά.

Ερωτήματα

1. Θεωρητικά αναμένουμε η εκτέλεση πριν από τον συγχρονισμό να είναι πιο γρήγορη από ότι μετά και για τις 2 μεθόδους. Αυτό συμβαίνει διότι πριν τον συγχρονισμό, οι αφαιρέσεις και οι προσθέσεις εκτελούνται παράλληλα από τα νήματα και έτσι γίνονται πολλές ταυτόχρονα και άρα ο χρόνος για την περάτωσή τους μειώνεται. Αυτή ακριβώς η παραλληλία δημιουργεί και πρόβλημα στην ορθή εκτέλεση του προγράμματος διότι δημιουργείται ανταγωνιστική συνθήκη στην πρόσβαση στην ίδια περιοχή μνήμης (εκεί που δείχνει ο pointer ip). Μόλις επεκτείνουμε τον κώδικα για να επιτύχουμε συγχρονισμό, όμως, ουσιαστικά καταργούμε αυτή την παραλληλία, αφού τόσο με τα atomic operations όσο και με τα mutexes κάθε νήμα περιμένει το άλλο να ολοκληρώσει το increment (ή το decrement) και μετά κάνει την δική της πρόσβαση στο σημείο της μνήμης που είναι αποθηκευμένη η μεταβλητή. Πράγματι, η θεωρητική αυτή προσδοκία μας επιβεβαιώνεται από τα αποτελέσματα της εντολής time.

```
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ±  time ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -6002052.
./simplesync-mutex  0.04s user 0.01s system 60% cpu 0.072 total
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ±  time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = -5632836.
./simplesync-atomic  0.05s user 0.00s system 95% cpu 0.054 total
```

```
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ±  time ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
./simplesync-mutex  1.01s user 0.33s system 141% cpu 0.946 total
orion@orionpc  ~/Desktop/ntua/semester6/operating_systems/lab/OSLab/exercise2/sync  main ±  time ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
./simplesync-atomic  0.51s user 0.00s system 161% cpu 0.318 total
```

2. Επιπλέον, παρατηρούμε πως μεταξύ των 2 μεθόδων συγχρονισμού, τα atomic operations είναι γρηγορότερα. Αυτό είναι λογικό, αφού τα atomic operations ουσιαστικά φροντίζουν απλά οι 3 εντολές assembly που αφορούν την ανάγνωση από τη μνήμη σε καταχωρητή, την αλλαγή του καταχωρητή και την εγγραφή πίσω στη μνήμη να γίνουν μαζί, ενώ τα mutexes λειτουργούν με spinlocks τα οποία έχουν πιο σύνθετη λειτουργία και καταναλώνεται επιπλέον χρόνος στα lock και unlock.

3. Παράγουμε το αρχείο με τον κώδικα assembly προσθέτοντας στο Makefile κώδικα:

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

   Με τη βοήθεια του αναγνωριστικού .loc 1 51 4 view που προσθέτει ο gcc με την βοήθεια του flag -g αντιλαμβανόμαστε πως η εντολή assembly που αντιστοιχεί στην εντολή __sync_add_and_fetch(ip, 1) είναι η lock addl $1, (%rbx). Αντίστοιχα, στην __sync_sub_and_fetch(ip, 1) έχουμε την εντολή assembly lock subl $1, (%rbx).

4. Παράγουμε το αρχείο με τον κώδικα assembly προσθέτοντας στο Makefile κώδικα:

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

   Με τη βοήθεια του αναγνωριστικού .loc 1 54 4 view που προσθέτει ο gcc με την βοήθεια του flag -g αντιλαμβανόμαστε πως η εντολή assembly που αντιστοιχεί στην εντολή pthread_mutex_lock(&mut) είναι η

   movq %r12, %rdi
   call pthread_mutex_lockPLT

   Αντίστοιχα, στην pthread_mutex_unlock(&mut) έχουμε την εντολή assembly call pthread_mutex_unlockPLT. Οι ίδιες εντολές καλούνται και στην αφαίρεση οπότε δεν έχει νόημα να τις ξαναδούμε.

2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

Ζητούμενα

- Καλούμαστε να επιλύσουμε το πρόβλημα συγχρονισμού NTHREADS νημάτων που τυπώνουν γραμμή γραμμή το Σύνολο Mandelbrot. Τροποποιούμε και επεικτείνουμε τον κώδικα mandel.c φτιάχοντας 2 διαφορετικά προγράμματα που το επιτυγχάνουν, ένα με σημαφόρους (mandel-semaphores.c) και ένα με μεταβλητές συνθήκης (mandel-condition-vars.c). Σε κάθε περίπτωση, το NTHREADS δίνεται από τον χρήστη ως όρισμα στη γραμμή εντολών.

- Πρέπει επιπλέον να φροντίσουμε ότι το τερματικό θα έχει το αρχικό χρώμα του καλώντας reset_xterm_color(1) για τον file descriptor 1 που αντιστοιχεί στο Standard Output. Όμοια στο τελευταίο ερώτημα κάνουμε το ίδιο στο sighandler που δημιουργούμε.

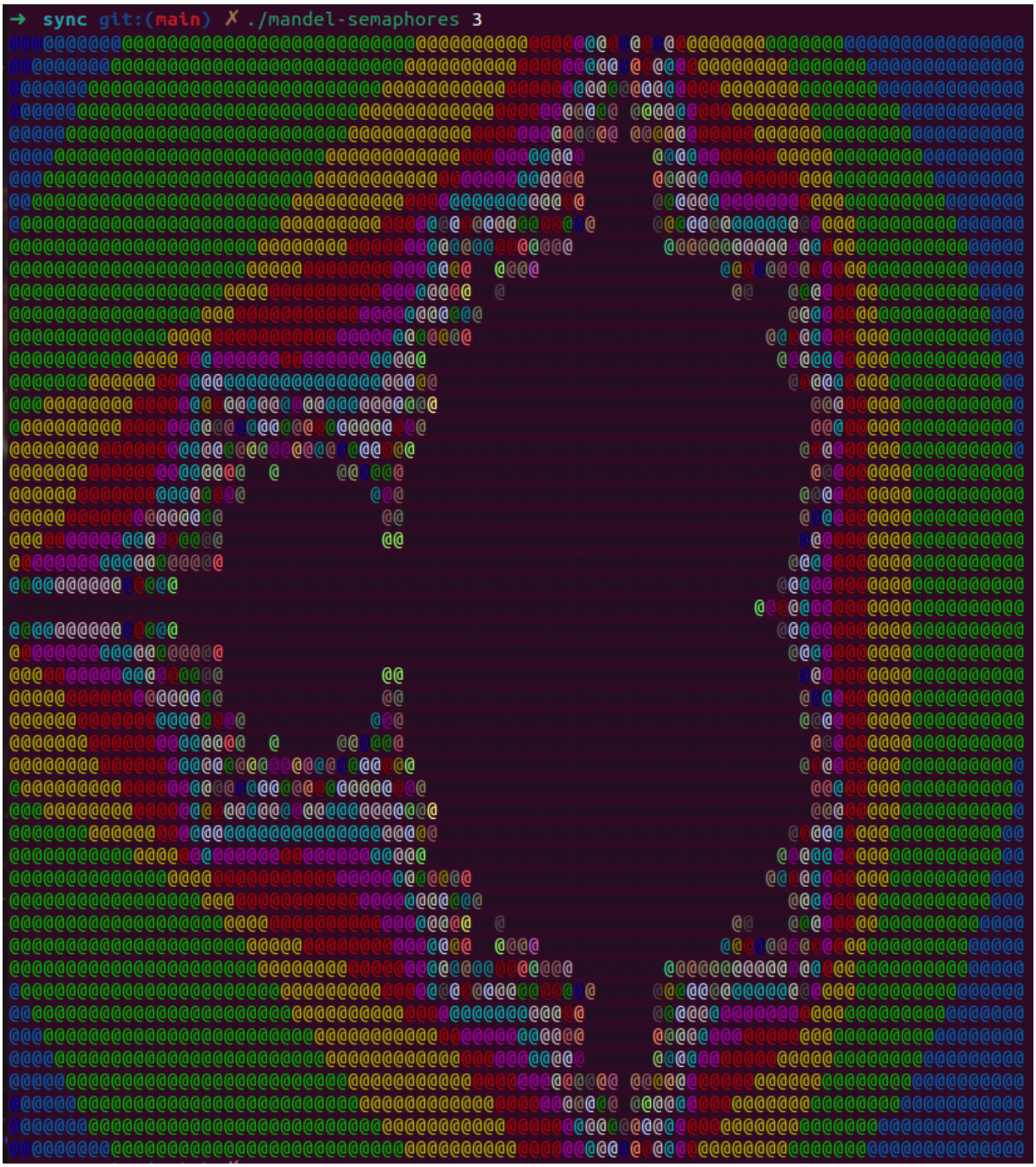

Figure 1: Συγχρονισμός των νημάτων με σημαφόρους

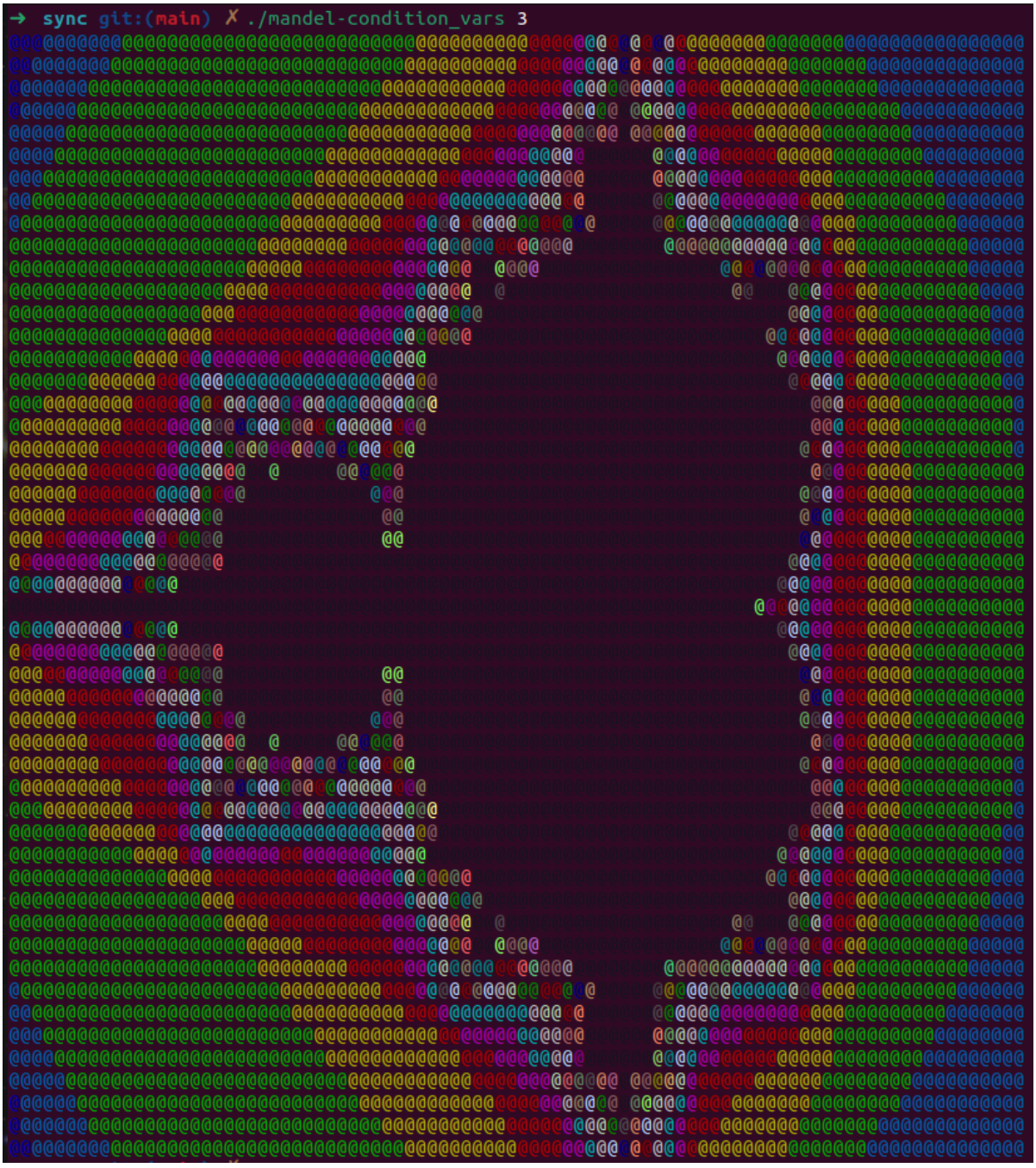

Figure 2: Συγχρονισμός των νημάτων με μεταβλητές συνθήκης

Ερωτήματα

1. Για να επιτύχουμε τον συγχρονισμό των NTHREADS νημάτων ώστε αυτά να τυπώνουν τις γραμμές στην οθόνη το ένα μετά το άλλο σε κυκλική σειρά, χρειαζόμαστε NTHREADS σημαφόρους. Αυτό συμβαίνει, διότι μόνο ένα νήμα μπορεί να βρίσκεται κάθε φορά στο κρίσιμο τμήμα, και αν είχαμε λιγότερους σημαφόρους δεν θα μπορούσαμε να γνωρίζουμε ποιό από όλα τα νήματα που περιμένουν στον ίδιο σημαφόρο πρέπει να μπει πρώτο. Θα δημιουργούνταν race conditions και θα χαλούσε ο συγχρονισμός του προγράμματος. Αντιθέτως, με NTHREADS σημαφόρους, το κάθε νήμα κάνει wait τον σημαφόρο που του αντιστοιχεί (τον κλειδώνει) και μόλις βγει από το κρίσιμο τμήμα κάνει post αυτόν που αντιστοιχεί στο επόμενο νήμα της κυκλικής σειράς (τον ξεκλειδώνει).

2. Με την εντολή time μετράμε τον χρόνο εκτέλεσης του σειριακού κώδικα mandel και των δύο παράλληλων προγραμμάτων για NTHREADS = 2. Παρακάτω παραθέτουμε τα αποτελέσματα. Σημειώνουμε πως έχει νόημα η σύγκριση, αφού τρέχουμε την εντολή σε σύστημα με τουλάχιστον 2 πυρήνες, όπως ελέγχθηκε και με την εντολή cat /proc/cpuinfo.

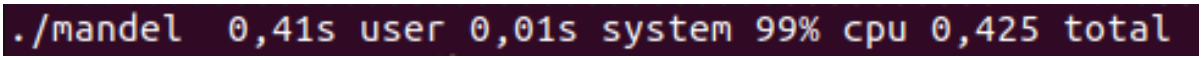

Figure 3: Χρόνος εκτέλεσης σειριακού κώδικα

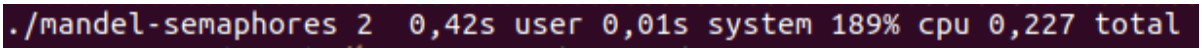

Figure 4: Χρόνος εκτέλεσης παραλληλοποιημένου κώδικα με σημαφόρους

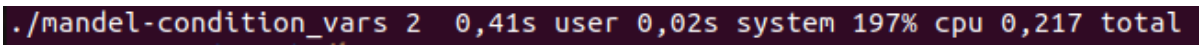

Figure 5: Χρόνος εκτέλεσης παραλληλοποιημένου κώδικα με μεταβλητές συνθήκης

Όπως βλέπουμε, το σειριακό πρόγραμμα χρειάστηκε συνολικά 0.425s, το παράλληλο με σημαφόρους 0.227s και το παράλληλο με μεταβλητές συνθήκης 0.217s.

3. Στη δεύτερη εκδοχή του προγράμματός μας χρησιμοποιήσαμε NTHREADS μεταβλητές συνθήκης. Με αυτόν τον τρόπο, το κάθε νήμα "ξυπνάει" μόνο όταν είναι πράγματι η δική του σειρά να εκτελεστεί. Θα μπορούσαμε να χρησιμοποιήσουμε και μόνο 1 μεταβλητή συνθήκης, έχοντας όλα τα νήματα να περιμένουν πάνω σε αυτή και το καθένα να την κάνει broadcast μόλις ολοκληρώσει το κρίσιμο τμήμα του. Αυτό θα σήμαινε πως κάθε φορά που ένα νήμα τελειώνε την εκτύπωση της γραμμής στην έξοδο και έστελνε το signal, θα "ξυπνούσαν" όλα τα νήματα και θα έλεγχαν μέσω του υπάρχοντος while loop (εκεί χρειαζόμαστε ακόμα τις NTHREADS boolean μεταβλητές για να ξέρουμε ποιανού νήματος είναι η σειρά) αν ήρθε η σειρά τους. Το νήμα του οποίου είναι η σειρά θα μπει στο κρίσιμο τμήμα του, ενώ όλα τα άλλα θα "ξανακοιμηθούν". Παρότι αυτό θα δούλευε και θα χρησιμοποιούσε λίγο λιγότερο χώρο, θα ήταν πιο αργό αφού θα χανόταν χρόνος στο αναίτιο ξύπνημα των NTHREADS-1 νημάτων και στο context switching μέχρι αυτά να ξανακοιμηθούν.

4. Τα παράλληλα προγράμματα που φτιάξαμε εμφανίζουν και τα δύο επιτάχυνση σε σχέση με το σειριακό, εκτελώμενα και τα δύο σχεδόν στον μισό συνολικό χρόνο, με το πρόγραμμα με μεταβλητές συνθήκης να είναι το γρηγορότερο. Παρότι η εκτύπωση των χαρακτήρων στην έξοδο (που αποτελεί και το κρίσιμο τμήμα) γίνεται πάντα σειριακά, η επιτάχυνση που παρατηρείται είναι αναμενόμενη, αφού ο υπολογισμός των γραμμών του Συνόλου Mandelbrot στα παράλληλα προγράμματα γίνεται παράλληλα, ταυτόχρονα στους δύο πυρήνες του επεξεργαστή (αφού αυτό δεν υπάρχει λόγος να ανήκει στο κρίσιμο τμήμα).

5. Πατώντας Ctrl+C όσο έτρεχε ο κώδικας, το τερματικό αφηνόταν στο χρώμα του τελευταίου χαρακτήρα που τυπώθηκε. Για να το αποφύγουμε αυτό, δημιουργήσαμε έναν signal handler για το σήμα SIGINT, ο οποίος φροντίζει προτού τερματίσει τον κώδικα να κάνει reset το χρώμα στο τερματικό στο αρχικό χρώμα του, χρησιμοποιώντας την συνάρτηση reset_xterm_color που δινόταν.

Παρακάτω ακολουθούν οι κώδικες των ζητουμένων των 2 ασκήσεων.

A Κώδικας άσκησης 1

```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

#if defined(SYNC_ATOMIC)
# define USE_ATOMIC_OPS 1
#else
# define USE_ATOMIC_OPS 0
#endif

pthread_mutex_t mut;

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 {
      if(pthread_mutex_lock(&mut) != 0) perror("Mutex lock error\n");
      /* You cannot modify the following line */
      ++(*ip);
      if(pthread_mutex_unlock(&mut) != 0) perror("Mutex unlock error\n");
    }
  }
  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 */
```

```
75        __sync_sub_and_fetch(ip, 1);
76        /* ... */
77      } else {
78        if(pthread_mutex_lock(&mut) != 0) perror("Mutex lock error\n");
79        /* You cannot modify the following line */
80        --(*ip);
81        if(pthread_mutex_unlock(&mut) != 0) perror("Mutex unlock error\n");
82      }
83    }
84    fprintf(stderr, "Done decreasing variable.\n");
85
86    return NULL;
87  }
88
89
90  int main(int argc, char *argv[])
91  {
92    int val, ret, ok;
93    pthread_t t1, t2;
94
95    /*
96     * Initial value
97     */
98    val = 0;
99
100   /*
101    * Create threads
102    */
103   ret = pthread_create(&t1, NULL, increase_fn, &val);
104   if (ret) {
105     perror_pthread(ret, "pthread_create");
106     exit(1);
107   }
108   ret = pthread_create(&t2, NULL, decrease_fn, &val);
109   if (ret) {
110     perror_pthread(ret, "pthread_create");
111     exit(1);
112   }
113
114   /*
115    * Wait for threads to terminate
116    */
117   ret = pthread_join(t1, NULL);
118   if (ret)
119     perror_pthread(ret, "pthread_join");
120   ret = pthread_join(t2, NULL);
121   if (ret)
122     perror_pthread(ret, "pthread_join");
123
124   /*
125    * Is everything OK?
126    */
127   ok = (val == 0);
128
129   printf("%sOK, val = %d.\n", ok ? "" : "NOT ", val);
130
131   return ok;
132 }
```

B Κώδικας άσκησης 2

B.1 Κώδικας με σημαφόρους

```c
/*
 * 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 <fcntl.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
*/
const int y_chars = 50;
const 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)
*/
const double xmin = -1.8, xmax = 1.0;
const 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 NTHREADS;

struct thread_info {
    pthread_t thread_id;
    int fd;
    int line;
};

sem_t *semaphores;

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;
}

/*
 * 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: int fd, int line
{
    struct thread_info *thread = (struct thread_info *) arg;
  int fd = thread->fd;
  int line = thread->line;
  /*
   * A temporary array, used to hold color values for the line being drawn
   */
  int color_val[x_chars];

  for( ; line < y_chars; line += NTHREADS) {
    compute_mandel_line(line, color_val);
    sem_wait(&semaphores[(line)%NTHREADS]); //lock current semaphore
    output_mandel_line(fd, color_val);
    sem_post(&semaphores[(line + 1)%NTHREADS]); //unlock the next semaphore
  }
  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 argument_handling(int argc, char **argv) {
  if(argc != 2) {
    perror("There should one argument: the number of threads wanted.\n");
    exit(1);
  }
  if(safe_atoi(argv[1], &NTHREADS) == -1){
    perror("atoi error!\n");
    exit(1);
  }
  if(NTHREADS <= 0){
    perror("The number of threads should be a positive integer.\n");
    exit(1);
  }
}

void sigintHandler(int sig_num) {
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }
  reset_xterm_color(1);
  exit(1);
}

int main(int argc, char **argv){
  argument_handling(argc, argv);
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }
  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.
   */

    struct thread_info *threads;

    threads = (struct thread_info *) safe_malloc(NTHREADS * sizeof(struct thread_info));
    semaphores = (sem_t*) safe_malloc(NTHREADS * sizeof(sem_t));

  sem_init(&semaphores[0], 0, 1); //initialize the 0th semaphore to 1

  for(int i = 1; i < NTHREADS; ++i) {
        int ret = sem_init(&semaphores[i], 0, 0); //and all else to 0
    if(ret) {
      perror("Semaphore init");
      exit(1);
    }
  }

    for(int i = 0; i < NTHREADS; ++i) {
    threads[i].fd = 1;
    threads[i].line = i;
        int ret = pthread_create(&threads[i].thread_id, NULL, compute_and_output_mandel_line, &threads[
    i]);
        if(ret) {
            perror("pthread_create");
      exit(1);
        }
    }

    for(int i = 0; i < NTHREADS; ++i) { //join all threads after their executions
        int ret = pthread_join(threads[i].thread_id, NULL);
        if(ret) {
```

```
            perror("pthread_join");
      exit(1);
        }
    }

  for (int i = 0; i < NTHREADS; ++i) { //destroy every semaphore
    int ret = sem_destroy(&semaphores[i]);
    if(ret) {
      perror("Semaphore destroy");
      exit(1);
    }
  }

  reset_xterm_color(1);
  free(threads);
  free(semaphores);
  return 0;
}
```

B.2 Κώδικας με μεταβλητές συνθήκης

```
/*
 * 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 <fcntl.h>
#include <stdbool.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
*/
const int y_chars = 50;
const 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)
*/
const double xmin = -1.8, xmax = 1.0;
const 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 NTHREADS;

struct thread_info {
    pthread_t thread_id;
    int fd;
    int line;
};

pthread_mutex_t mut;
```

```
pthread_cond_t *conds;
bool *myturn;

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;
}

/*
 * 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: int fd, int line
{
```

```
    struct thread_info *thread = (struct thread_info *) arg;
  int fd = thread->fd;
  int line = thread->line;
  /*
   * A temporary array, used to hold color values for the line being drawn
   */
  int color_val[x_chars];

  for( ; line < y_chars; line += NTHREADS) {

    compute_mandel_line(line, color_val);
    pthread_mutex_lock(&mut); //lock mutex
    // wait until previous thread sends signal
    while(!myturn[line%NTHREADS]) pthread_cond_wait(&conds[line%NTHREADS], &mut);
    myturn[line%NTHREADS] = false;
    output_mandel_line(fd, color_val);
    myturn[(line+1)%NTHREADS] = true;
        pthread_cond_signal(&conds[(line+1)%NTHREADS]); //send signal to next thread
    pthread_mutex_unlock(&mut); //unlock mutex
  }
  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 argument_handling(int argc, char **argv) {
  if(argc != 2) {
    perror("There should one argument: the number of threads wanted.\n");
    exit(1);
  }
  if(safe_atoi(argv[1], &NTHREADS) == -1){
    perror("atoi error!\n");
    exit(1);
  }
  if(NTHREADS <= 0){
    perror("The number of threads should be a positive integer.\n");
    exit(1);
  }
}

void sigintHandler(int sig_num) {
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }
  reset_xterm_color(1);
  exit(1);
}

int main(int argc, char **argv){
  argument_handling(argc, argv);
  if(signal(SIGINT, sigintHandler) < 0){
    perror("Could not establish SIGINT handler");
    exit(1);
  }
  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.
   */

    struct thread_info *threads;

    threads = (struct thread_info *) safe_malloc(NTHREADS * sizeof(struct thread_info));
  conds = (pthread_cond_t* ) safe_malloc(NTHREADS * sizeof(pthread_cond_t));
```

```
  myturn = (bool *) safe_malloc(NTHREADS * sizeof(bool));

  myturn[0] = true;

    for(int i = 0; i < NTHREADS; ++i) {
    threads[i].fd = 1;
    threads[i].line = i;
        int ret = pthread_create(&threads[i].thread_id, NULL, compute_and_output_mandel_line, &threads[
    i]);
        if(ret) {
            perror("pthread_create");
      exit(1);
        }
    }

    for(int i = 0; i < NTHREADS; ++i) { //join all threads after their executions
        int ret = pthread_join(threads[i].thread_id, NULL);
        if(ret) {
            perror("pthread_join");
      exit(1);
        }
    }

  reset_xterm_color(1);
  free(threads);
  free(conds);
  return 0;
}
```