ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Λειτουργικά Συστήματα

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2020-2021

Ονοματεπώνυμο	Αριθμός Μητρώου
Γουλόπουλος Δημήτριος	03107627
Σιαφάκας Ξενοφών	03115753

oslaba116

Άσκηση 3: Συγχρονισμός

Πηγαίος Κώδικας: simplesync.c

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

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

#define N 10000000

#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 cnt_mutex;
```

```
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)
        {
            __sync_add_and_fetch(ip, 1);
        }
        else
        {
            pthread_mutex_lock(&cnt_mutex);
            ++(*ip);
            pthread_mutex_unlock(&cnt_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)
        {
            __sync_add_and_fetch(ip, -1);
        }
        else
        {
            pthread_mutex_lock(&cnt_mutex);
            --(*ip);
            pthread_mutex_unlock(&cnt_mutex);
        }
    }
    fprintf(stderr, "Done decreasing variable.\n");
    return NULL;
}
```

```
int main(int argc, char *argv[])
{
    int val, ret, ok;
    pthread_t t1, t2;
    val = 0;
    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);
    }
    ret = pthread_join(t1, NULL);
    if (ret)
        perror_pthread(ret, "pthread_join");
    ret = pthread_join(t2, NULL);
    if (ret)
        perror_pthread(ret, "pthread_join");
    ok = (val == 0);
    printf("%sOK, val = %d.\n", ok ? "" : "NOT ", val);
    return ok;
}
```

Χρησιμοποιήστε το παρεχόμενο Makefile για να μεταγλωττίσετε και να τρέξετε το πρόγραμμα. Τι παρατηρείτε; Γιατί;

```
$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 2891619.
```

```
$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
NOT OK, val = 8192350.
```

Παρατηρούμε ότι η τιμή της `val` στο τέλος δεν είναι 0.

Αυτό συμβαίνει επειδή οι εντολές σε γλώσσα μηχανής για την ανάγνωση-αύξηση-αποθήκευση της τιμής της μεταβλητής `val` δεν είναι `atomic`.

• Επεκτείνετε τον κώδικα του simplesync.c ώστε η εκτέλεση των δύο νημάτων στο εκτελέσιμο simplesync-mutex να συγχρονίζεται με χρήση POSIX mutexes. Επιβεβαιώστε την ορθή λειτουργία του προγράμματος.

• Επεκτείνετε τον κώδικα του simplesync.c ώστε η εκτέλεση των δύο νημάτων στο εκτελέσιμο simplesync-atomic να συγχρονίζεται με χρήση ατομικών λειτουργιών του GCC. Επιβεβαιώστε την ορθή λειτουργία του προγράμματος.

Η έξοδος του προγράμματος στα δύο εκτελέσιμα προγράμματα είναι:

```
~/00--askisi--3$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
```

```
~/00--askisi--3$ ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
```

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;

```
$ 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    0m0.412s
user    0m0.812s
sys     0m0.000s

$ time ./simplesync-mutex
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
real    0m3.671s
user    0m3.660s
sys     0m2.904s
```

Η εκτέλεση του εκτελέσιμου προγράμματος χωρίς συγχρονισμό είναι πιο γρήγορη καθώς δεν απαιτείται κάποιος περιορισμός στην εκτέλεση των δύο threads.
Και στις 2 περιπτώσεις ήταν:

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

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

Η μέθοδος με χρήση ατομικών λειτουργιών είναι αισθητά γρηγορότερη

Τα atomic operations μεταφράζονται σε μια εντολή assembly.

Η υλοποίηση με mutexes αποτελεί μια πιο high-level προσσέγγιση,

η οποία ενσωματώνει την έννοια των atomic operations για να υλοποιηθεί.

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.

Παρακάτω παρατέθεται ο κώδικας του __sync_fetch_and_add μεταφραζμένος σε assembly:

```
.L3:
.loc 1 50 0
movq -16(%rbp), %rax
lock addl $1, (%rax)
.loc 1 47 0
addl $1, -4(%rbp)
```

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.

```
.L3:
.loc 1 54 0
movl $cnt_mutex, %edi
call pthread_mutex_lock
.loc 1 55 0
movq -16(%rbp), %rax
movl (%rax), %eax
leal 1(%rax), %edx
movq -16(%rbp), %rax
movl %edx, (%rax)
.loc 1 56 0
movl $cnt_mutex, %edi
call pthread_mutex_unlock
.loc 1 47 0
addl $1, -4(%rbp)
```

Πηγαίος Κώδικας: mandel.c

```
#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

typedef struct
{
      pthread_t tid;
      int line;
      sem_t mutex;
} newThread_t;

newThread_t *thread;

int y_chars = 50;
int x_chars = 90;

double xmin = -1.8, xmax = 1.0;
double ymin = -1.0, ymax = 1.0;

double xstep;
double ystep;
int NTHREADS;
```

```
void compute_mandel_line(int line, int color_val[])
{
    double x, y;
    int n;
    int val;
    y = ymax - ystep * line;
    for (x = xmin, n = 0; n < x_chars; x += xstep, n++)
    {
        val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
        if (val > 255)            val = 255;
        val = xterm_color(val);
        color_val[n] = val;
    }
}

void output_mandel_line(int fd, int color_val[])
{
    int i;
    char point = '@';
    char newline = '\n';
    for (i = 0; i < x_chars; i++)
    {
        set_xterm_color(fd, color_val[i]);
        if (write(fd, &point, 1) != 1)
        {
            perror("compute_and_output_mandel_line: write point");
            exit(1);
        }
    }
    if (write(fd, &newline, 1) != 1)
    {
        perror("compute_and_output_mandel_line: write newline");
        exit(1);
    }
}
```

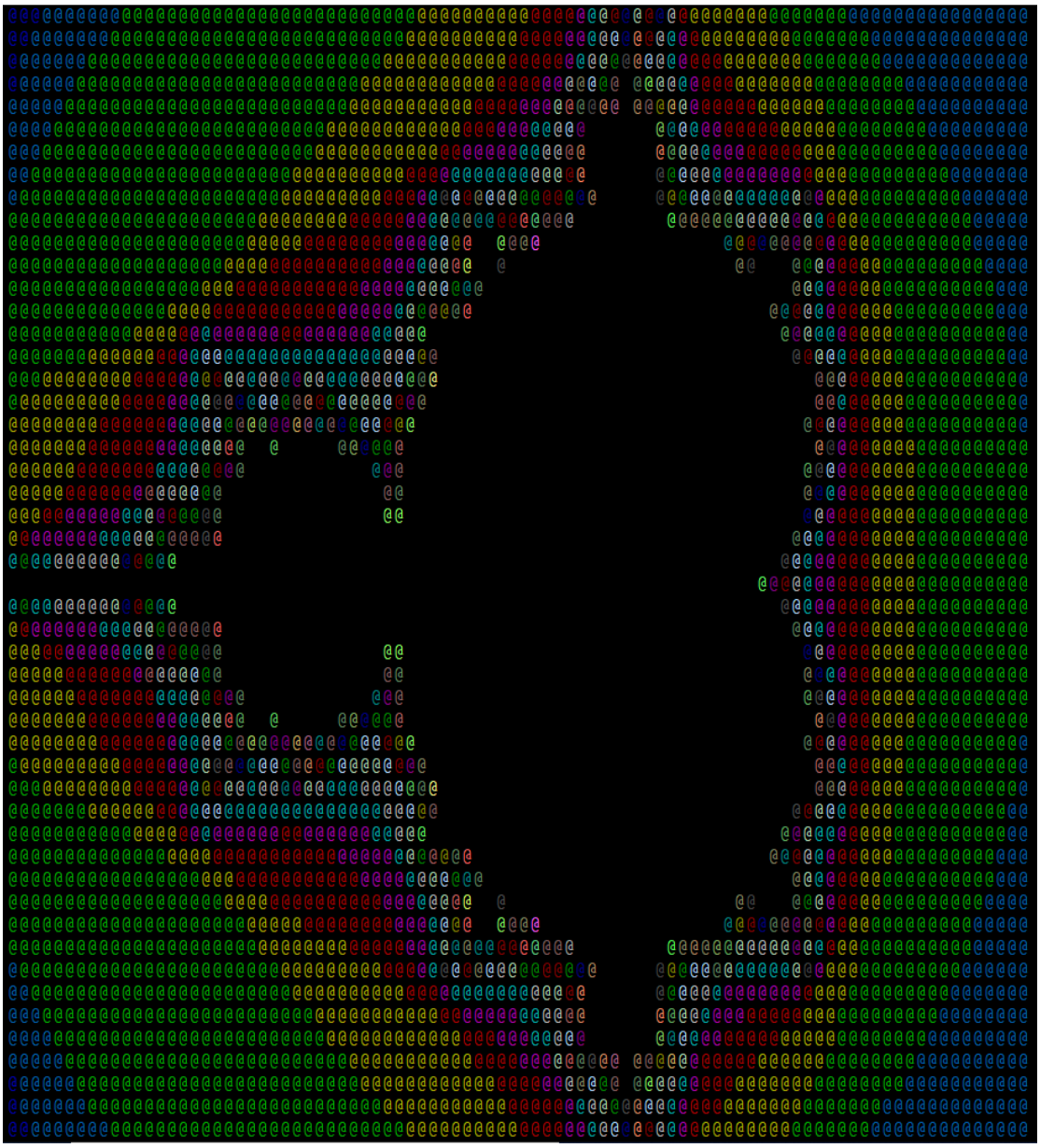
```
void *compute_and_output_mandel_line(void *arg)
{
    int line = *(int*) arg;
    int index;
    int color_val[x_chars];
    for (index = line; index < y_chars; index += NTHREADS)
    {
        compute_mandel_line(index, color_val);
        sem_wait(&thread[(index % NTHREADS)].mutex);
        output_mandel_line(STDOUT_FILENO, color_val);
        sem_post(&thread[((index % NTHREADS) + 1) % NTHREADS].mutex);
    }
    return NULL;
}


void unexpectedSignal(int sign)
{
    signal(sign, SIG_IGN);
    reset_xterm_color(1);
    exit(1);
}
```

```
int main(void)
{
    int line;
    signal(SIGINT, unexpectedSignal);
    xstep = (xmax - xmin) / x_chars;
    ystep = (ymax - ymin) / y_chars;
    printf("Enter Number of Threads: ");
    scanf("%d", &NTHREADS);
    if (NTHREADS < 1 || NTHREADS > y_chars)
    {
        printf("Input is not valid\n");
        return 0;
    }
    thread = (newThread_t*) malloc(NTHREADS* sizeof(newThread_t));
    if (thread == NULL)
    {
        perror("");
        exit(1);
    }
    if ((sem_init(&thread[0].mutex, 0, 1)) == -1)
    {
      // The sem_init function returns 0 on success and -1 on error
        perror("");
        exit(1);
    }
    for (line = 1; line < NTHREADS; line++)
    {
        if ((sem_init(&thread[line].mutex, 0, 0)) == -1)
        {
            perror("");
            exit(1);
        }
    }
```

```
    for (line = 0; line < NTHREADS; line++)
    {
        thread[line].line = line;
        if ((pthread_create(&thread[line].tid, NULL,
compute_and_output_mandel_line, &thread[line].line)) != 0)
        {
            perror("");
            exit(1);
        }
    }
    for (line = 0; line < NTHREADS; line++)
    {
        if ((pthread_join(thread[line].tid, NULL)) != 0)
        {
            perror("");
            exit(1);
        }
    }
    for (line = 0; line < NTHREADS; line++)
    {
        sem_destroy(&thread[line].mutex);
    }
    free(thread);
    reset_xterm_color(1);
    return 0;
}
```

Έξοδος εκτέλεσης προγράμματος:

Η έξοδος του προγράμματος από το εκτελέσιμο πρόγραμμα παρατίθεται στην παρακάτω εικόνα:



Ερωτήσεις:

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

Για την υλοποίηση χρησιμοποιούνται **N** σημαφόροι, ένα σημαφόρο για το κάθε **thread**. Για την ακρίβεια, ο σημαφόρος **i** ενεργοποιείται με την λήξη της εργασιάσς του **i**-1 νήματος και σηματοδοτεί την έναρξη του **N**+1 νήματος.

Πιο αναλυτικά το σχήμα συγχρονισμού του προγράμματος λειτουργεί σαν κυκλικός δακτύλιος, με το κάθε νήμα να παίρνει την άδεια προς την εκτέλεση από το προηγούμενό του.

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.

Οι μετρήσεις είναι οι παρακάτω:

```
~/00--askisi--3$  time (echo 6 | ./mandel)
Enter Number of Threads:
real    0m0.226s
user    0m1.004s
sys     0m0.016s
```

```
~/00--askisi--3$  time (echo 1 | ./mandel)
Enter Number of Threads:
real    0m1.032s
user    0m0.992s
sys     0m0.016s
```

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;

Παρατηρώντας τις παραπάνω μετρήσεις το παράλληλο πρόγραμμα εμφανίζει επιτάχυνση δεδομένου, ότι η φάση υπολογισμού γίνεται παράλληλα και ο συγχρονισμός επιτυγχάνεται κατά την διάρκεια του τυπώματος, ώστε να γίνει με την σωστή σειρά.

Σε άλλη περίπτωση δεν θα είχαμε την επιτάχυνση καθώς το κάθε νήμα θα περίμενε το προηγούμενο για να υπολογιστεί, με αποτέλεσμα το πρόγραμμα να εκφυλίζεται στο αντίστοιχο σειριακό.

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;

Με το πάτημα Ctrl+C, το πρόγραμμα τερματίζει καθώς του στέλνεται σήμα SIGINT, με αποτέλεσμα το χρώμα των γραμμών του τερματικού παραμένει σε αυτό που είχε επιλεχθεί για τύπωμα ακριβώς πριν τερματιστεί.

Δημιουργώντας τον δικό μας sig-handler που με το πάτημα Ctrl+C, επαναφέρει το χρώμα του τερματικού πριν τερμαστεί.

Οι εντολές που χρησιμοποιούνται είναι οι παρακάτω:

```
signal(SIGINT, unexpectedSignal);
```

όπου unexpectedSignal είναι μια συνάρτηση:

```
void unexpectedSignal(int sign)
{
    signal(sign, SIG_IGN);
    reset_xterm_color(1);
    exit(1);
}
```