3^{η} Αναφορά στα Λειτουργικά Συστήματα

Ιωάννης Αλεξόπουλος (03117001)
Αλέξανδρος Κυριακάκης (03112163)

June 6, 2020

1 Άσκηση 1

Κάναμε χρήση της εντολής make για το δοθέν Makefile οπότε έγινε η μεταγλώττιση και η σύνδεση των αρχείων κώδικα που περιείχε, το καθένα με τις συγκεκριμένες προδιαγραφές που καθόριζε το Makefile.

Παρατηρήσαμε ότι παρότι κανείς θα περίμενε το τελικό αποτέλεσμα της εκτέλεσης του παραχθέντος simplesync να είναι 0, το αποτέλεσμα ήταν αρκετά διαφορετικό από εκτέλεση σε εκτέλεση. Αυτό βέβαια είναι απολύτως λογικό αν σκεφτούμε ότι αναθέτοντας τη λειτουργία αύξησης της val κατά ένα στο ένα νήμα και τη λειτουργία μείωσης της val κατά ένα στο άλλο νήμα, αυτό έχει αποτέλεσμα η εκ περιτροπής χρήση του επεξεργαστή από τα νήματα να οδηγεί σε ατελή πρόσθεση ή αφαίρεση κατά ένα. Πιο συγκεκριμένα η εντολή πρόσθεσης(αντίστοιχα αφαίρεσης) μεταφράζεται σε περισσότερες της μίας εντολής σε assembly, οπότε αν έχουμε αλλαγή του νήματος που απασχολεί τον επεξεργαστή εντός της περιόδου που είμαστε σε αυτό το κρίσιμο κομμάτι κώδικα τότε η λειτουργία δε θα ολοκληρωθεί επιτυχώς και τα αποτελέσματα θα είναι απρόβλεπτα. Φαίνεται λοιπόν η ανάγκη να θωρακίσουμε με κάποιο τρόπο τα κρίσιμα αυτά κομμάτια κώδικα.

Έπειτα βλέπουμε ότι από το ίδιο αρχείο κώδικα παράγονται δύο διαφορετικά εκτελέσιμα simplesync-atomic και simplesync-mutex. Αυτό είναι απολύτως λογικό αν δούμε το περιεχόμενο του Makefile. Συγκεκριμένα με τις εντολές -D SYNC_ATOMIC και έπειτα -D SYNC_MUTEX κάνουμε define, καθορίζουμε ποια θα είναι η τιμή του USE_ATOMIC_OPS εντός του simplesync.c που λειτουργεί σαν flag για το αν θα χρησιμοποιηθούν mutexes ή atomic operations (το συμπεραίνουμε από τη χρήση των δομών if-else όπου η συνθήκη είναι αυτό το flag). Επομένως με δύο διαφορετικές εντολές μεταγλώττισης εντός του Makefile έχουμε δύο διαφορετικά εκτελέσιμα.

1.1 Ερώτημα 1

```
$ time ./simplesync-mutex

real    0m1.231s
user    0m1.596s
sys     0m0.766s

$ time ./simplesync-atomic

real    0m0.132s
user    0m0.257s
sys     0m0.000s
```

Με χρήση της εντολής time(1) παρατηρούμε ότι και στις δύο περιπτώσεις οι χρόνοι εκτέλεσης των εκτελέσιμων που λειτουργούν με συγχρονισμό είναι αυξημένοι σε σχέση με τον χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό. Αυτό συμβαίνει καθώς μόνο ένα νήμα βρίσκεται στο κρίσιμο τμήμα της αύξησης ή μείωσης του μετρητή σε κάθε χρονική στιγμή, σε αντίθεση με το αρχικό πρόγραμμα όπου δεν υπάρχει αυστηρός έλεγχος σχετικό με το ποιο νήμα θα κάνει χρήση των υπολογιστικών πόρων του συστήματος.

1.1.1 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)
```

```
24
25  #define N 10000000
26
27  /* Dots indicate lines where you are free to insert code at will
    ↪  */
28  /* ... */
29
30  #if defined(SYNC_ATOMIC) ^ defined(SYNC_MUTEX) == 0
31  # error You must #define exactly one of SYNC_ATOMIC or
    ↪  SYNC_MUTEX.
32  #endif
33
34  #if defined(SYNC_ATOMIC)
35  # define USE_ATOMIC_OPS 1
36  #else
37  # define USE_ATOMIC_OPS 0
38  #endif
39  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
40  void *increase_fn(void *arg)
41  {
42   int i;
43   volatile int *ip = arg;
44
45   fprintf(stderr, "About to increase variable %d times\n", N);
46   for (i = 0; i < N; i++) {
47    if (USE_ATOMIC_OPS) {
48              __sync_fetch_and_add(&ip, 1);
49    } else {
50     pthread_mutex_lock(&mutex1);
51     /* You cannot modify the following line */
52     ++(*ip);
53     pthread_mutex_unlock(&mutex1);
54    }
55   }
56   fprintf(stderr, "Done increasing variable.\n");
57
58   return NULL;
59  }
60
61  void *decrease_fn(void *arg)
62  {
63   int i;
64   volatile int *ip = arg;
65
66   fprintf(stderr, "About to decrease variable %d times\n", N);
67   for (i = 0; i < N; i++) {
```

```
68    if (USE_ATOMIC_OPS) {
69     /* ... */
70     /* You can modify the following line */
71               __sync_fetch_and_add(&ip, -1);
72     /* ... */
73    } else {
74     pthread_mutex_lock(&mutex1);
75     /* You cannot modify the following line */
76     --(*ip);
77     pthread_mutex_unlock(&mutex1);
78    }
79   }
80   fprintf(stderr, "Done decreasing variable.\n");
81
82   return NULL;
83  }
84
85
86  int main(int argc, char *argv[])
87  {
88   int val, ret, ok;
89   pthread_t t1, t2;
90
91   /*
92    * Initial value
93    */
94   val = 0;
95
96   /*
97    * Create threads
98    */
99   ret = pthread_create(&t1, NULL, increase_fn, &val);
100  if (ret) {
101   perror_pthread(ret, "pthread_create");
102   exit(1);
103  }
104  ret = pthread_create(&t2, NULL, decrease_fn, &val);
105  if (ret) {
106   perror_pthread(ret, "pthread_create");
107   exit(1);
108  }
109
110  /*
111   * Wait for threads to terminate
112   */
113  ret = pthread_join(t1, NULL);
```

```
114   if (ret)
115    perror_pthread(ret, "pthread_join");
116   ret = pthread_join(t2, NULL);
117   if (ret)
118    perror_pthread(ret, "pthread_join");
119
120   /*
121    * Is everything OK?
122    */
123   ok = (val == 0);
124
125   printf("%sOK, val = %d.\n", ok ? "" : "NOT ", val);
126
127   return ok;
128  }
```

1.2 Ερώτημα 2

Παρατηρούμε ότι μεταξύ της υλοποίησης με mutexes και της υλοποίησης με atomic operations η δεύτερη είναι σημαντικά πιο γρήγορη. Αυτό είναι λογικό μιας και η δεύτερη υλοποίηση είναι πιο χαμηλού επιπέδου, αφού χρησιμοποιεί συγκεκριμένες εντολές του επεξεργαστή χωρίς να τίθεται κάποιο νήμα σε κατάσταση sleep, σε αντίθεση τη διαδικασία κλείδωμα-ξεκλείδωμα του mutex, η οποία έχει επιπλέον χρονικό κόστος από το λειτουργικό σύστημα κατά τις κλήσεις sleep-wake του νήματος που περιμένει να εισέλθει στο κρίσιμο τμήμα. Αυτό που θέλει προσοχή σε αυτό το σημείο είναι ότι υπάρχει μόνο μια μικρή γκάμα atomic operations και άρα για την πλειονότητα των εφαρμογών αναγκαστικά καταφεύγουμε σε άλλες μεθόδους συγχρονισμού.

1.3 Ερώτημα 3

Παράγοντας τα κατάλληλα αρχεία που περιέχουν τον assembly κώδικα που αντιστοιχεί στο πρόγραμμά μας.

Για το thread που έχει αναλάβει την αύξηση έχουμε:

```
.LBE17:
    .loc 1 47 0
    lock addq   $1, (%rsp)
```

Για το thread που έχει αναλάβει την μείωση έχουμε:

```
.LBE25:
    .loc 1 74 0
    lock subq   $1, (%rsp)
```

1.4 Ερώτημα 4

Κατά αναλογία με πριν έχουμε για το pthread_mutex_lock() τον παρακάτω assembly κώδικα:

```
.LBE17:
    .loc 1 52 0
    movq    %rbp, %rdi
    call    pthread_mutex_lock@PLT
```

Αντίστοιχα για το pthread_mutex_unlock() έχουμε:

```
    .loc 1 55
    call    pthread_mutex_unlock@PLT
```

2 Άσκηση 2

2.1 Ερώτημα 1

2.1.1 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 <semaphore.h>
#include <signal.h>
#include "mandel-lib.h"
#include <pthread.h>

#define MANDEL_MAX_ITERATION 100000

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

/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
```

```c
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;
/*
 * SIGINT (CTRL + C) handler
 */
void sigint_handler (int signum)
{
    reset_xterm_color(1);
    exit(1);
}
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 */
 int thrid; /* Application-defined thread id */
 int thrcnt;
};

int safe_atoi(char *s, int *val)
{
 long l;
 char *endp;

 l = strtol(s, &endp, 10);
 if (s != endp && *endp == '\0') {
  *val = l;
```

```
73    return 0;
74   } else
75    return -1;
76  }
77
78  void *safe_malloc(size_t size)
79  {
80   void *p;
81
82   if ((p = malloc(size)) == NULL) {
83    fprintf(stderr, "Out of memory, failed to allocate %zd
      ↪  bytes\n",
84     size);
85    exit(1);
86   }
87
88   return p;
89  }
90
91  void usage(char *argv0)
92  {
93   fprintf(stderr, "Usage: %s thread_count array_size\n\n"
94    "Exactly one argument required:\n"
95    "    thread_count: The number of threads to create.\n",
96    argv0);
97   exit(1);
98  }
99
100
101
102 /*
103  * This function computes a line of output
104  * as an array of x_char color values.
105  */
106 void compute_mandel_line(int line, int color_val[])
107 {
108  /*
109   * x and y traverse the complex plane.
110   */
111  double x, y;
112
113  int n;
114  int val;
115
116  /* Find out the y value corresponding to this line */
117  y = ymax - ystep * line;
```

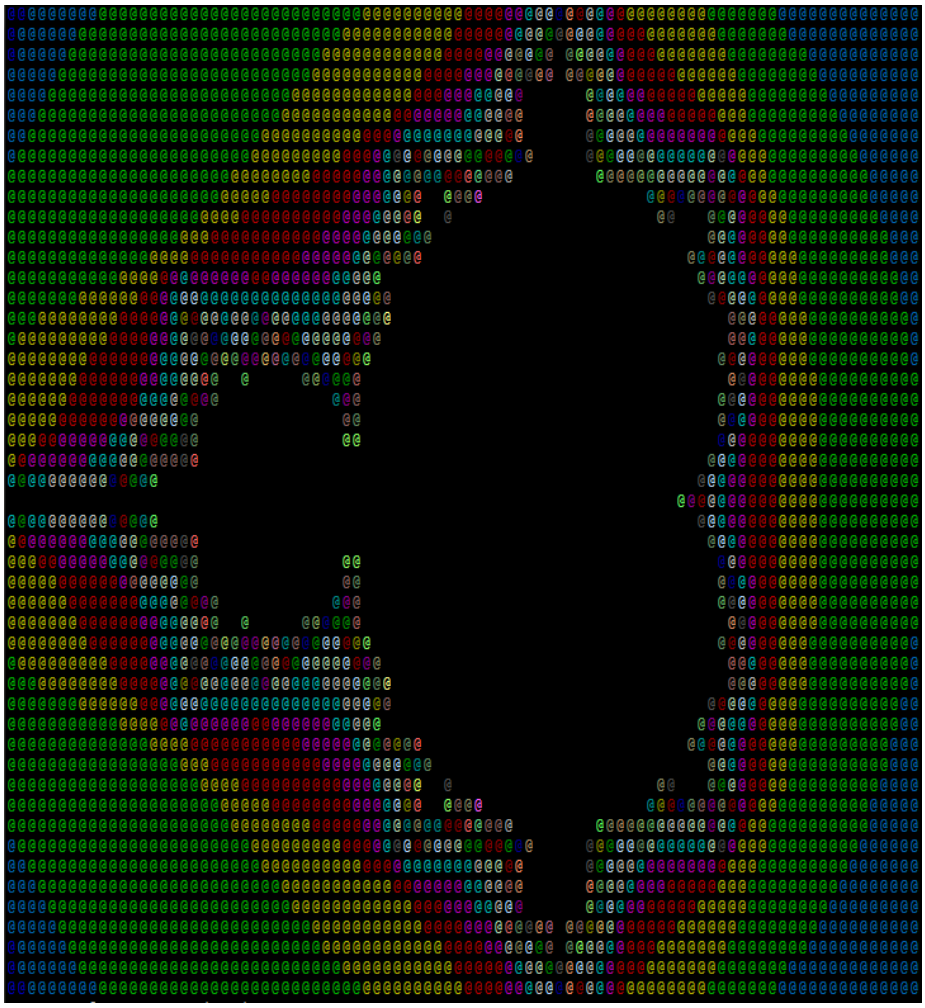
```
118
119   /* and iterate for all points on this line */
120   for (x = xmin, n = 0; n < x_chars; x+= xstep, n++) {
121
122     /* Compute the point's color value */
123     val = mandel_iterations_at_point(x, y, MANDEL_MAX_ITERATION);
124     if (val > 255)
125       val = 255;
126
127     /* And store it in the color_val[] array */
128     val = xterm_color(val);
129     color_val[n] = val;
130   }
131 }
132
133 /*
134  * This function outputs an array of x_char color values
135  * to a 256-color xterm.
136  */
137 void output_mandel_line(int fd, int color_val[])
138 {
139   int i;
140
141   char point ='@';
142   char newline='\n';
143
144   for (i = 0; i < x_chars; i++) {
145     /* Set the current color, then output the point */
146     set_xterm_color(fd, color_val[i]);
147     if (write(fd, &point, 1) != 1) {
148       perror("compute_and_output_mandel_line: write point");
149       exit(1);
150     }
151   }
152
153   /* Now that the line is done, output a newline character */
154   if (write(fd, &newline, 1) != 1) {
155     perror("compute_and_output_mandel_line: write newline");
156     exit(1);
157   }
158 }
159
160 void *compute_and_output_mandel_line(void *arg)
161 {
162   /* The line will be given from threads
```

```
163     * A temporary array, used to hold color values for the line
    ↪  being drawn
164     */
165    struct thread_info_struct *thr = arg;
166    int i;
167
168    for (i = thr->thrid; i < y_chars; i += thr->thrcnt){
169     compute_mandel_line(i, thr->color_val);
170     sem_wait(&mutex[i % thr->thrcnt]);
171     output_mandel_line(1, thr->color_val);
172     sem_post(&mutex[(i+1) % thr->thrcnt]);
173    }
174
175
176    return NULL;
177
178
179   }
180
181   int main(int argc, char *argv[])
182   {
183    xstep = (xmax - xmin) / x_chars;
184    ystep = (ymax - ymin) / y_chars;
185
186    /*
187     * draw the Mandelbrot Set, one line at a time.
188     * Output is sent to file descriptor '1', i.e., standard output.
189     */
190
191    int i, ret, thrcnt;
192    struct thread_info_struct *thr;
193
194    /*
195     * Parse the command line
196     */
197    if (argc != 2)
198     usage(argv[0]);
199    if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0) {
200     fprintf(stderr, "`%s' is not valid for `thread_count'\n",
      ↪  argv[1]);
201     exit(1);
202    }
203       struct sigaction sa;
204       sigset_t sigset;
205       sa.sa_handler = sigint_handler;
206       sa.sa_flags = SA_RESTART;
```

```
207       sigemptyset(&sigset);
208       sa.sa_mask = sigset;
209       if (sigaction(SIGINT, &sa, NULL) < 0) {
210           perror("sigaction");
211           exit(1);
212       }
213     thr = safe_malloc(thrcnt * sizeof(*thr));
214     mutex = safe_malloc(thrcnt * sizeof(sem_t));
215     for (i = 0; i < thrcnt; i++) {
216      /* Initialize per-thread structure */
217      thr[i].thrid = i;
218      thr[i].thrcnt = thrcnt;
219      thr[i].color_val = safe_malloc(x_chars * sizeof(int));
220      (i == 0) ?  sem_init(&mutex[i],0,1) : sem_init(&mutex[i],0,0);
221      /* Spawn new thread */
222      ret = pthread_create(&thr[i].tid,
       ↪  NULL,compute_and_output_mandel_line, &thr[i]);
223      if (ret) {
224     //  perror_pthread(ret, "pthread_create");
225        exit(1);
226      }
227     }
228
229     /*
230      * Wait for all threads to terminate
231      */
232     for (i = 0; i < thrcnt; i++) {
233      ret = pthread_join(thr[i].tid, NULL);
234      if (ret) {
235     //  perror_pthread(ret, "pthread_join");
236        exit(1);
237      }
238     }
239
240
241
242
243     //for (line = 0; line < y_chars; line++) {
244     // compute_and_output_mandel_line(1, line);
245     //}
246       for (i = 0; i < thrcnt; i++) {
247        sem_destroy(&mutex[i]);
248       }
249     reset_xterm_color(1);
250     return 0;
251   }
```

Από τον παραπάνω κώδικα βλέπουμε ότι έχουμε κάνει χρήση τόσων σημαφόρων όσο είναι το πλήθος των νημάτων.

2.1.2 Ενδεικτική έξοδος εκτέλεσης με 10 νήματα



2.2 Ερώτημα 2

Για το σειριακό πρόγραμμα με την κατάλληλη χρήση της time έχουμε ως αποτέλεσμα:

```
real    0m0.534s
user    0m0.522s
sys     0m0.012s
```

Για το παράλληλο πρόγραμμα με δύο νήματα πάλι με τη χρήση της time έχουμε:

```
real    0m0.294s
user    0m0.556s
sys     0m0.016s
```

Βλέπουμε δηλαδή ότι ο χρόνος παράλληλο πρόγραμμα με 2 νήματα έχουμε σχεδόν το μισό πραγματικό χρόνο και το μισό χρόνο συστήματος, ενώ ο χρόνος που αντιλαμβάνεται ο χρήστης είναι σχεδόν ίδιος.

Με την εκτέλεση της εντολής cat /proc/cpuinfo παίρνουμε μια πληθώρα στοιχείων για τον "υπολογιστή μας". Κοιτώντας προσεκτικά βλέπουμε σε μια γραμμή "cpu cores : 4" πράγμα που προφανώς σημαίνει ότι το υπολογιστικό μηχάνημά μας έχει 4 πυρήνες.

2.3 Ερώτημα 3

Το πρόγραμμά μας αποκρίνεται με εμφανώς καλύτερη ταχύτητα και αυτό γιατί έχουμε φροντίσει να γίνονται μαζεμένα οι υπολογισμοί των γραμμών και μετά την εμφάνιση αυτών. Το κρίσιμο κομμάτι όπως καταλαβαίνουμε είναι κυρίως η εκτύπωση κάθε στοιχείου έκαστης γραμμής. Αν φροντίζαμε κάθε γραμμή να υπολογίζεται από ένα νήμα και να τυπώνεται και μόνο έπειτα να αναλαμβάνει τον έλεγχο ένα άλλο νήμα για μια άλλη γραμμή τότε η απόδοση δε θα βελτιωνόταν καθόλου. Θα ήταν σαν να γινόταν σειριακά

2.4 Ερώτημα 4

Αρχικά πατώντας Ctrl-C εν μέσω της διαδικασίας, πριν δηλαδή ολοκληρωθεί η εκτέλεση του προγράμματος τότε δεν έχουν γίνει reset τα χρώματα και το τερματικό εμφανίζει ότι γράφουμε με χρώμα. Αυτό το αντιμετωπίσαμε κάνοντας ένα reset εντός του sigint_handler.