3η Ομάδα Ασκήσεων

ΣΤΑ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ομάδα Α38:

Ιωακειμίδη Αθηνά

Α.Μ.: 03114758

Μαυρομμάτης Ιάσων

Α.Μ.: 03114771

Εξάμηνο 7ο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Άσκηση 1

Πηγαίος κώδικας των συναρτήσεων:

```
pthread_mutex_t count_lock;

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(&count_lock);
			++(*ip);
			pthread_mutex_unlock(&count_lock);
		}
	}
	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_sub_and_fetch(ip, 1);
		} else {
			pthread_mutex_lock(&count_lock);
			--(*ip);
			pthread_mutex_unlock(&count_lock);
		}
	}
	fprintf(stderr, "Done decreasing variable.\n");

	return NULL;
}
```

(Η main παραμένει ίδια με το αρχικό αρχείο και γι' αυτό δεν παρατίθεται.)

Έξοδος εκτέλεσης:

Για το εκτελέσιμο που κάνει χρήση ατομικών λειτουργιών του GCC:

```
oslaba38@orion:~/exer3/ask1$ ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
```

Για το εκτελέσιμο που κάνει χρήση POSIX mutexes:

```
oslaba38@orion:~/exer3/ask1$ ./simplesync-mutex
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.
```

Απαντήσεις στις ερωτήσεις:

1. Οι έξοδοι εκτέλεσης του time(1) είναι οι παρακάτω:

```
oslaba38@orion:~/exer3/ask1$ time ./simplesync-atomic
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.

real    0m0.979s
user    0m1.840s
sys     0m0.000s
```

```
oslaba38@orion:~/exer3/ask1$ 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    0m3.340s
user    0m3.488s
sys     0m2.612s
```

   Παρατηρούμε ότι σε σχέση με τον χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό (που δεν παρατίθεται εδώ) υπάρχει μια μικρή καθυστέρηση. Αυτό συμβαίνει λόγω του ότι υπάρχει κρίσιμο τμήμα στο οποίο έχει πρόσβαση ένα μόνο thread τη φορά. Η καθυστέρηση δημιουργείται όταν το ένα thread αναγκάζεται να περιμένει το άλλο.
2. Από τις εξόδους εκτέλεσης του time βλέπουμε ότι η χρήση ατομικών λειτουργιών είναι γρηγορότερη από τη χρήση POSIX mutexes. Αυτό συμβαίνει γιατί στην πρώτη περίπτωση αρκεί να γράψουμε μία εντολή και όλη η δουλειά θα γίνει σε χαμηλότερο επίπεδο, ενώ στην δεύτερη περίπτωση (POSIX mutexes) απαιτείται η χρήση πολλαπλών εντολών για την υλοποίηση του ίδιου πράγματος.
3. Για την αύξηση του val η εντολή: `__sync_add_and_fetch(ip, 1);`
   μεταφράζεται στις παρακάτω εντολές assembly:

```
.L2:
        .loc 1 49 0
        lock addl       $1, (%rbx)
```

4. Επίσης για την αύξηση του val, οι εντολές:

```
pthread_mutex_lock(&count_lock);
++(*ip);
pthread_mutex_unlock(&count_lock);
```

μεταφράζονται στις εξής εντολές assembly:

```
.L2:
        .loc 1 51 0
        movl    $count_lock, %edi
        call    pthread_mutex_lock
.LVL4:
        .loc 1 52 0
        movl    0(%rbp), %eax
        .loc 1 53 0
        movl    $count_lock, %edi
        .loc 1 52 0
        addl    $1, %eax
        movl    %eax, 0(%rbp)
        .loc 1 53 0
        call    pthread_mutex_unlock
```

Άσκηση 2

Πηγαίος κώδικας:

Προσθέσαμε τις παρακάτω βιβλιοθήκες και κάναμε το παρακάτω define:

```
#include <errno.h>
#include <pthread.h>
#include <stdint.h>
#include <semaphore.h>

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

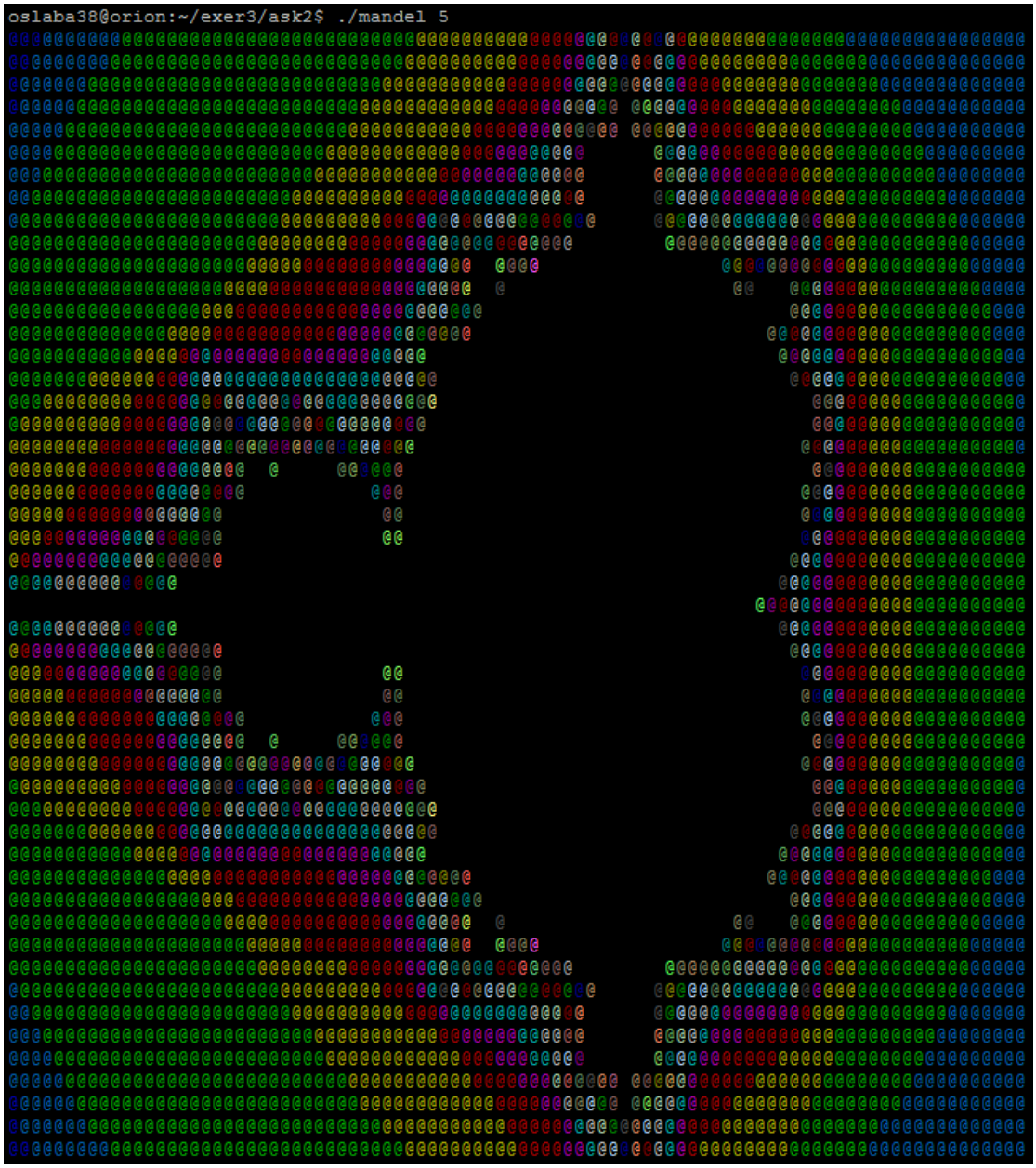
(Οι συναρτήσεις που παρεμβάλλονται εδώ παραμένουν ίδιες με το αρχικό αρχείο και γι' αυτό δεν παρατίθενται.)

```
int n;
sem_t *sem;

void *compute_and_output_mandel_line(void *arg)
{
      int i, j, color_val[x_chars];
        j = *((int *) arg);

      for (i = j; i<y_chars; i+=n) {
            compute_mandel_line(i, color_val);

            sem_wait(&sem[j]);
            output_mandel_line(1, color_val);
            sem_post(&sem[(j+1) % n]);
      }

      return NULL;
}

int main(int argc, char *argv[])
{
      if (argc!=2) {
            perror("one argument expected");
            exit(1);
      }

      n = atoi(argv[1]);

      int i, ret, i2[n];
      pthread_t t[n];
      sem=malloc(sizeof(sem_t)*n);

      sem_init(&sem[0], 0, 1);
      for (i=1; i<n; i++) sem_init(&sem[i], 0, 0);

      xstep = (xmax - xmin) / x_chars;
      ystep = (ymax - ymin) / y_chars;

      for (i=0; i<n; i++) {
            i2[i]=i;
            ret = pthread_create(&t[i], NULL,
                        compute_and_output_mandel_line, i2+i);
            if (ret) {
                  perror_pthread(ret, "pthread_create");
```

```
                exit(1);
            }
        }

        for (i=0; i<n; i++) {
            ret = pthread_join(t[i], NULL);
            if (ret) perror_pthread(ret, "pthread_join");
        }

        for (i=0; i<n; i++) sem_destroy(&sem[i]);

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

Έξοδος εκτέλεσης:

Η έξοδος της εκτέλεσης παραμένει ίδια ό,τι είσοδο και να δώσουμε.



Απαντήσεις στις ερωτήσεις:

1. Για το σχήμα συγχρονισμού που υλοποιούμε χρειάζονται n σημαφόροι, δηλαδή ένας σημαφόρος ανά mutex.
2. Χρησιμοποιήσαμε λάπτοπ με επεξεργαστή δύο πυρήνων και βρήκαμε:

   `time ./mandel 1` (σειριακός υπολογισμός)

   ```
   real    0m0.957s
   user    0m0.932s
   sys     0m0.020s
   ```

   `time ./mandel 2` (παράλληλος υπολογισμός)

   ```
   real    0m0.673s
   user    0m0.910s
   sys     0m0.027s
   ```
3. Το παράλληλο πρόγραμμά μας εμφανίζει επιτάχυνση στην εκτέλεσή του αλλά όχι στην έξοδό του. Στη δική μας υλοποίηση το κρίσιμο τμήμα περιλαμβάνει μόνο την εκτύπωση της εξόδου για να εξασφαλίσει ότι η εκτύπωση θα γίνει σειριακά και ότι η εικόνα θα εκτυπωθεί σωστά. Έτσι η εκτύπωση είναι συγχρονισμένη αλλά ο υπολογισμός της κάθε γραμμής γίνεται παράλληλα στα threads μας.
4. Εάν πατήσουμε Ctrl-C κατά τη διάρκεια της εκτέλεσης του προγράμματος παρατηρούμε ότι τα επόμενα γράμματα διατηρούν το τελευταίο χρώμα που χρησιμοποιήθηκε κατά την εκτέλεση της εικόνας.
   Για να εξασφαλίσουμε ότι το τερματικό θα επαναφέρεται στην προηγούμενη κατάστασή του φτιάχνουμε έναν signal handler:

   ```
   //στη main: signal(SIGINT, reset);
   void reset(int sign)
   {
         reset_xterm_color(1);
         exit(1);
   }
   ```

Άσκηση 3

Πηγαίος κώδικας:

Στο αρχικό αρχείο κάναμε τις παρακάτω αλλαγές:

```
struct kgarten_struct {
	int vt;
	int vc;
	int ratio;

	pthread_mutex_t mutex;
	pthread_cond_t cond;
};

void child_enter(struct thread_info_struct *thr)
{
	if (!thr->is_child) {
		fprintf(stderr, "Internal error: %s called for a Teacher
					thread.\n", __func__);
		exit(1);
	}

	pthread_mutex_lock(&thr->kg->mutex);
	while (thr->kg->vc+1 > thr->kg->vt * thr->kg->ratio)
		pthread_cond_wait(&thr->kg->cond, &thr->kg->mutex);
	++(thr->kg->vc);
	pthread_mutex_unlock(&thr->kg->mutex);

	fprintf(stderr, "THREAD %d: CHILD ENTER\n", thr->thrid);
}

void child_exit(struct thread_info_struct *thr)
{

	if (!thr->is_child) {
		fprintf(stderr, "Internal error: %s called for a Teacher
					thread.\n", __func__);
		exit(1);
	}

	fprintf(stderr, "THREAD %d: CHILD EXIT\n", thr->thrid);

	pthread_mutex_lock(&thr->kg->mutex);
	--(thr->kg->vc);
	pthread_cond_broadcast(&thr->kg->cond);
	pthread_mutex_unlock(&thr->kg->mutex);
}

void teacher_enter(struct thread_info_struct *thr)
{
	if (thr->is_child) {
		fprintf(stderr, "Internal error: %s called for a Child
					thread.\n", __func__);
		exit(1);
	}

	fprintf(stderr, "THREAD %d: TEACHER ENTER\n", thr->thrid);

	pthread_mutex_lock(&thr->kg->mutex);
	++(thr->kg->vt);
	pthread_cond_broadcast(&thr->kg->cond);
```

```
	pthread_mutex_unlock(&thr->kg->mutex);
}

void teacher_exit(struct thread_info_struct *thr)
{
	if (thr->is_child) {
		fprintf(stderr, "Internal error: %s called for a Child
					thread.\n", __func__);
		exit(1);
	}

	pthread_mutex_lock(&thr->kg->mutex);
	while (thr->kg->vc > (thr->kg->vt-1) * thr->kg->ratio)
		pthread_cond_wait(&thr->kg->cond, &thr->kg->mutex);
	--(thr->kg->vt);
	pthread_mutex_unlock(&thr->kg->mutex);

	fprintf(stderr, "THREAD %d: TEACHER EXIT\n", thr->thrid);
}
```

Απαντήσεις στις ερωτήσεις:

1. Το σχήμα συγχρονισμού μας μας εξασφαλίζει μόνο ότι διατηρείται η σωστή αναλογία δασκάλων και παιδιών ώστε να υπάρχει σωστή επίβλεψη, αλλά η εξυπηρέτηση των νημάτων γίνεται τυχαία. Έτσι υπάρχει περίπτωση καθώς ένας δάσκαλος να περιμένει να φύγει, να υπάρχουν παιδιά που να μπαίνουν στον παιδότοπο.
2. Στην υλοποίησή μας για κάθε δάσκαλο που επιχειρεί να φύγει ή παιδί που επιχειρεί να μπει στον παιδότοπο γίνεται έλεγχος ότι η αναλογία τους είναι η ζητούμενη. Άρα μπορεί να εμφανιστεί κατάσταση συναγωνισμού στο κομμάτι του ελέγχου αυτού. Εμείς για να εξασφαλίσουμε ότι θα γίνει ορθός έλεγχος της συνθήκης την ορίσαμε ως κρίσιμο τμήμα.

Έξοδος εκτέλεσης:

```
oslaba38@orion:~/exer3/ask3$ ./kgarten 10 7 2
Thread 2 of 10. START.
Thread 2 [Child]: Entering.
Thread 0 of 10. START.
Thread 0 [Child]: Entering.
Thread 3 of 10. START.
Thread 3 [Child]: Entering.
Thread 1 of 10. START.
Thread 8 of 10. START.
Thread 5 of 10. START.
Thread 5 [Child]: Entering.
Thread 6 of 10. START.
Thread 6 [Child]: Entering.
Thread 7 of 10. START.
Thread 4 of 10. START.
Thread 4 [Child]: Entering.
Thread 1 [Child]: Entering.
Thread 9 of 10. START.
Thread 9 [Teacher]: Entering.
THREAD 9: TEACHER ENTER
Thread 8 [Teacher]: Entering.
THREAD 8: TEACHER ENTER
Thread 8 [Teacher]: Entered.
THREAD 2: CHILD ENTER
Thread 2 [Child]: Entered.
            Thread 2: Teachers: 2, Children: 4
THREAD 0: CHILD ENTER
Thread 0 [Child]: Entered.
            Thread 0: Teachers: 2, Children: 4
Thread 7 [Teacher]: Entering.
THREAD 7: TEACHER ENTER
THREAD 4: CHILD ENTER
Thread 4 [Child]: Entered.
            Thread 4: Teachers: 3, Children: 5
THREAD 1: CHILD ENTER
Thread 1 [Child]: Entered.
            Thread 1: Teachers: 3, Children: 6
Thread 9 [Teacher]: Entered.
            Thread 9: Teachers: 3, Children: 6
THREAD 3: CHILD ENTER
Thread 3 [Child]: Entered.
THREAD 5: CHILD ENTER
Thread 5 [Child]: Entered.
```

Η έξοδος του προγράμματος αυτού συνεχίζεται έτσι επ’ άπειρον.