Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Λειτουργικά Συστήματα

Άσκηση 3: Συγχρονισμός

1.1 Συγχρονισμός σε υπάρχοντα κώδικα

```
/*
 * 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 new_mutex;
#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) {
			__sync_fetch_and_add(ip, 1);
		} else {
			pthread_mutex_lock(&new_mutex);
			/* You cannot modify the following line */
			++(*ip);
			pthread_mutex_unlock(&new_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_sub_and_fetch(ip, 1);
		} else {
			pthread_mutex_lock(&new_mutex);
			/* You cannot modify the following line */
			--(*ip);
			pthread_mutex_unlock(&new_mutex);
		}
	}
	fprintf(stderr, "Done decreasing variable.\n");

	return NULL;
}



int main(int argc, char *argv[])
{
	int val, ret, ok;
	pthread_t t1, t2;
	/*
	 * 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);

	return ok;
}
```

Χρησιμοποιήσαμε το παρεχόμενο Makefile για να μεταγλωττίσουμε και να τρέξουμε το πρόγραμμα. Παρατηρήσαμε ότι για κάθε αρχείο .c χρησιμοποιείται η παράμετρος -pthread άρα ο κώδικας χρησιμοποιεί νήματα (threads). Τέλος παρατηρήσαμε πως το αρχείο Makefile εκτελεί δύο μεταγλωττίσεις για το αρχείο simplesync.c και παράγει δύο εκτελέσιμα . Το ένα παράγεται με τη παράμετρο -DSYNC_MUTEX και ένα με τη παράμετρο -DSYNC_ATOMIC. Και τα δύο χρησιμοποιούν τον κώδικα simplesync.c για να πράξουν ένα object file το καθένα. Από τα object files φτιάχνονται τα εκτελέσιμα simplesync-atomic και simplesync-mutex.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου:

- simplesync-mutex

```
oslaba05@os-node1:~/Final$ ./simplesync-mutex
About to decrease variable 10000000 times
About to increase variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
```

- simplesync-atomic

```
oslaba05@os-node1:~/Final$ ./simplesync-atomic
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.
```

Απαντήσεις:

1. Παρατηρούμε ότι στη περίπτωση όπου έχουμε συγχρονισμό ο χρόνος που απαιτείται για να εκτελεστεί το πρόγραμμα είναι μεγαλύτερος από τη περίπτωση που δεν έχουμε συγχρονισμό. Αυτό οφείλεται στο γεγονός ότι με την ύπαρξη του συγχρονισμού για να εκτελέσει την λειτουργία ένα νήμα πρέπει να περιμένει να πάρει το κλειδί από το άλλο νήμα και άρα δεν εκτελούνται ταυτόχρονα οι υπολογισμοί, άρα χρειαζόμαστε περισσότερο χρόνο.

2. Η μέθοδος ατομικών λειτουργιών είναι πιο γρήγορη από τη μέθοδο POSIX mutexes. Αυτό οφείλεται στο γεγονός ότι με POSIX mutexes οι διεργασίες αναστέλλουν την λειτουργίας τους όσο το κύριο τμήμα είναι κατειλημμένο. Οι ατομικές λειτουργίες όμως προσπαθούν να εκτελεστούν συνέχεια μέχρι να τα καταφέρουν . Επίσης στις ατομικές λειτουργίες το κρίσιμο τμήμα είναι μικρότερο

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

3. Η χρήση ατομικών λειτουργιών του GCC μεταγλωττίζεται σε

lock addl $1, (%eax) για την πρόσθεση και σε

lock subl $1, (%eax) για την αφαίρεση.

Αυτό το βλέπω με την χρήση των κάτωθεν εντολών :

gcc -O2 -S -g -DSYNC_ATOMIC -c simplesync.c

gcc -c -g -Wa,-a,-ad -DSYNC_ATOMIC simplesync.c > atomic.lst

```
46:simplesync.c  ****         if (USE_ATOMIC_OPS) {
47:simplesync.c  ****             __sync_fetch_and_add(ip, 1);
40                     .loc 1 47 0
41 003b 488B45F8       movq    -8(%rbp), %rax
42 003f F0830001       lock addl   $1, (%rax)
45:simplesync.c  ****     for (i = 0; i < N; i++) {
43                     .loc 1 45 0
44 0043 8345F401       addl    $1, -12(%rbp)
45                  .L2:
45:simplesync.c  ****     for (i = 0; i < N; i++) {
46                     .loc 1 45 0 is_stmt 0 discriminator 1
47 0047 817DF47F       cmpl    $9999999, -12(%rbp)
47       969800
48 004e 7EEB           jle .L3
```

```
 67:simplesync.c  ****         if (USE_ATOMIC_OPS) {
 68:simplesync.c  ****             __sync_sub_and_fetch(ip, 1);
 99                     .loc 1 68 0
100 00b0 488B45F8      movq    -8(%rbp), %rax
101 00b4 F0832801      lock subl   $1, (%rax)
 66:simplesync.c  ****     for (i = 0; i < N; i++) {
102                    .loc 1 66 0
103 00b8 8345F401      addl    $1, -12(%rbp)
104                 .L6:
 66:simplesync.c  ****     for (i = 0; i < N; i++) {
105                    .loc 1 66 0 is_stmt 0 discriminator 1
106 00bc 817DF47F      cmpl    $9999999, -12(%rbp)
106      969800
107 00c3 7EEB          jle .L7
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

4. Με την χρήση των κάτωθεν εντολών :

gcc -O2 -S -g -DSYNC_MUTEX -c simplesync.c

gcc -c -g -Wa,-a,-ad -DSYNC_MUTEX simplesync.c > mutex.lst

βλέπουμε την πιο κάτω μεταγλώττιση :

```
48:simplesync.c  ****         } else {
49:simplesync.c  ****             pthread_mutex_lock(&new_mutex);
40                       .loc 1 49 0
41 003b BF000000         movl    $new_mutex, %edi
41      00
42 0040 E8000000         call    pthread_mutex_lock
42      00
50:simplesync.c  ****             /* You cannot modify the following line */
51:simplesync.c  ****             ++(*ip);
43                       .loc 1 51 0
```

```
 69:simplesync.c  ****         } else {
 70:simplesync.c  ****             pthread_mutex_lock(&new_mutex);
108                      .loc 1 70 0
109 00cb BF000000        movl    $new_mutex, %edi
109      00
110 00d0 E8000000        call    pthread_mutex_lock
110      00
 71:simplesync.c  ****             /* You cannot modify the following line */
 72:simplesync.c  ****             --(*ip);
111                      .loc 1 72 0
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

1.2 Παράλληλος υπολογισμός του συνόλου Mandelbrot

```
/*
 * mandel.c
 *
 * A program to draw the Mandelbrot Set on a 256-color xterm.
 *
 */

#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <semaphore.h>
#include <pthread.h>

#include "mandel-lib.h"

#define MANDEL_MAX_ITERATION 100000

#define perror_pthread(ret, msg) \
```

```
        do { errno = ret; perror(msg); } while (0)


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


/*
 * Output at the terminal is is x_chars wide by y_chars long
*/
int y_chars = 50;
int x_chars = 90;
sem_t *sem;
int nthreads;
/*
 * 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.
 */


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 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[line][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 line)
{
        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[line][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 *thread_start_fn(void *arg)
{
   int line,*x = arg;

   int *color_val[y_chars];
   for (line =0; line < y_chars; line++)
      color_val[line] = (int *)malloc(x_chars*sizeof(int));

   for (line = *x; line < y_chars; line +=nthreads) {
          compute_mandel_line(line, color_val);
   }
      /*
      * If the semaphore's value is greater than zero, then the decrement
proceeds,
      * and the function returns, immediately. If the semaphore currently has the
value
```

```
        * zero, then the call blocks until either it becomes possible to perform the
decrement
        */
    for (line = *x; line < y_chars; line +=nthreads) {
        sem_wait(&sem[*x]);


            output_mandel_line(1, color_val,line);


        /* increments (unlocks) the semaphore pointed to by sem. */
        if (*x == nthreads-1) sem_post(&sem[0]);
        else sem_post(&sem[*x + 1]);
    }
    return NULL;
}




int main(int argc, char *argv[])
{
        int ret,i;
    /*
         * Parse the command line
         */
```

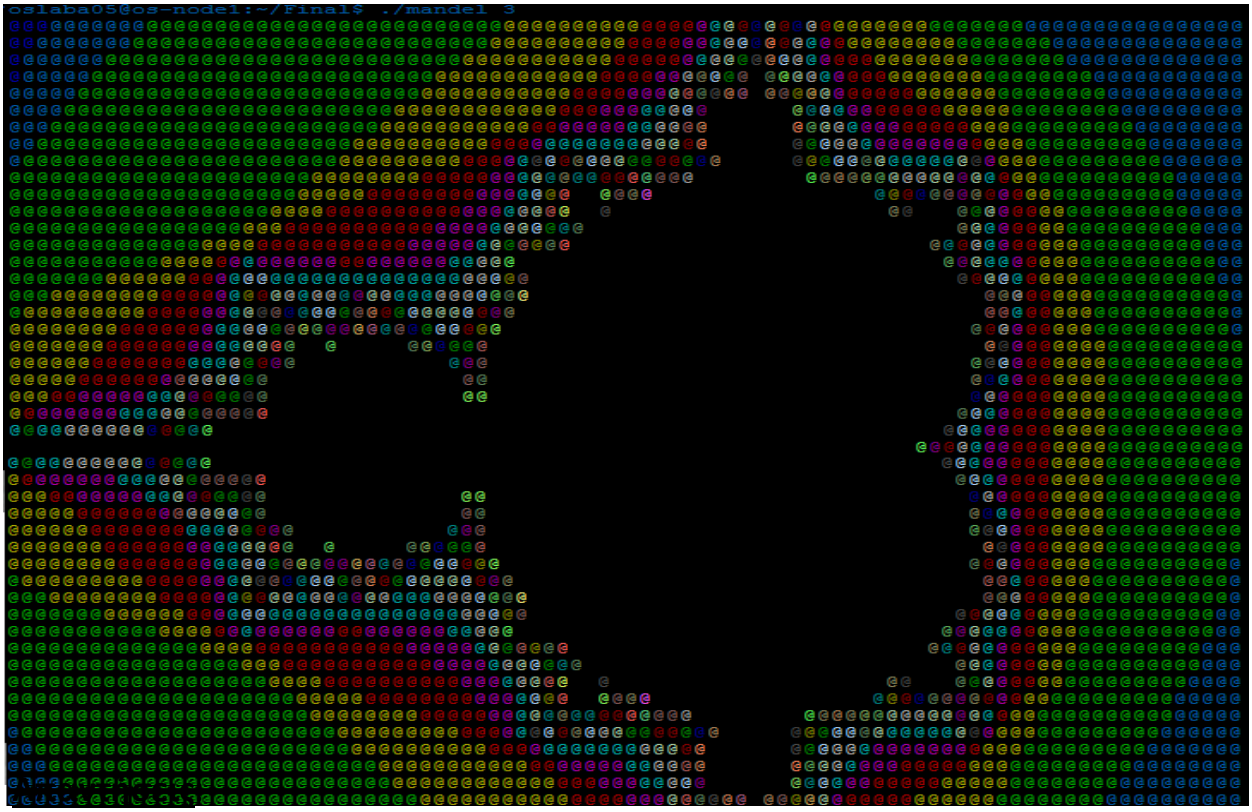
```
if (argc != 2){
   fprintf(stderr, "Usage: %s NTHREADS \n\n"
              "Exactly one argument required:\n"
              "    NTHREADS: The number of threads.\n\n",
              argv[0]);
         exit(1);
}
if (safe_atoi(argv[1], &nthreads) < 0 || nthreads <= 0) {
              fprintf(stderr, "`%s' is not valid for `nthreads'\n", argv[1]);
              exit(1);
      }
/*
       * Create threads
       */
pthread_t *tid;
tid =  malloc(nthreads*sizeof(pthread_t));

int *id;
id =  malloc(nthreads*sizeof(int));

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

```
        * Create semaphores
        */
   sem = malloc(nthreads*sizeof(sem_t));
   /*
   * initializes the unnamed semaphore at the address pointed to by sem.
   * int sem_init(sem_t *sem, int pshared, unsigned int value);
   * pshared = 0, then the semaphore is shared between the threads of a process
   * The value argument specifies the initial value for the semaphore.
   */
   sem_init(&sem[0],0,1);                          // semaphore 0 : allows 1 thread
inside
        for(i = 1; i<nthreads; i++ )
      sem_init(&sem[i],0,0);      // other semaphores allow 0 threads inside

   /* Spawn new thread */
   for (i = 0; i < nthreads; i++) {
      id[i] = i ;
            ret = pthread_create(&tid[i], NULL,thread_start_fn,&id[i]);
            if (ret) {
                  perror_pthread(ret, "pthread_create");
                  exit(1);
      }
   }
```

```
        /*
         * Wait for all threads to terminate
         */
        for (i = 0; i < nthreads; i++) {
                ret = pthread_join(tid[i], NULL);
                if (ret) {
                        perror_pthread(ret, "pthread_join");
                        exit(1);
                }
        }
   /*
         * Destroy all semaphores
         */
        for(i = 0; i<nthreads; i++ )
      sem_destroy(&sem[i]);

        return 0;
}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου



1. Για το σχήμα συγχρονισμού που υλοποιήσαμε χρειάστηκαν τόσοι σημαφόροι όσα είναι και τα νήματα, ώστε να μπορούμε να ελέγχουμε πότε θα έχει πρόσβαση το κάθε νήμα στο κρίσιμο τμήμα. Αυτό έχει νόημα καθώς τα νήματα μπορούν να εκτελούνται παράλληλα. Έτσι όταν ένα νήμα θέλει να χρησιμοποιήσει κάποιο κοινό πόρο μπαίνει στη διαδικασία να κλειδώσει μειώνοντας το σημαφόρο του και παίρνοντας έτσι τον έλεγχο. Με το τρόπο αυτό αποτρέπει σε άλλα νήματα να κάνουν την ίδια κίνηση εώς ότου αυτό να δώσει τον έλεγχο στον σημαφόρο του επόμενου στη σειρά νήματος.

2. Για την σειριακή εκτέλεση:

```
real    0m1.022s
user    0m0.988s
sys     0m0.008s
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Για την παράλληλη εκτέλεση:

```
real    0m0.511s
user    0m0.976s
sys     0m0.012s
```

3. Από τα πιο πάνω αποτελέσματα συμπεραίνουμε πως η παράλληλη εκτέλεση απαιτεί λιγότερο χρόνο από τη σειριακή. Αυτό εξηγείται εύκολα αφού το κάθε νήμα δεν περιμένει να τελειώσει το προηγούμενό του για να ξεκινήσει. Το κρίσιμο τμήμα είναι μικρό επειδή αποτελείται μόνο από τη φάση εξόδου της κάθε γραμμής, αφού ο υπολογισμός της κάθε γραμμής δεν εξαρτάται από τον υπολογισμό της προηγούμενης γραμμής. Έτσι και γι' αυτό το λόγο υπάρχει μια επιτάχυνση στο χρόνο εκτέλεσης .

4. Όταν πατάμε Ctrl-C ενώ το πρόγραμμα εκτελείται, αυτό τερματίζει. Επομένως η αλλαγή των χρωμάτων των χαρακτήρων δεν επαναφέρεται και παραμένει το χρώμα που τέθηκε τελευταίο με αποτέλεσμα να παραμένει χρωματισμένο το terminal. Θα μπορούσαμε να αντιμετωπίσουμε το πρόβλημα τη συνάρτηση χειρισμού σήματος sighandler() στην οποία θα υπάρχουν οι εντολές reset_xterm_color(1) και η exit(1). Έτσι με το πάτημα του Ctrl-C, θα ενεργοποιείται η sighandler() και θα επαναφέρει το αρχικό χρώμα πριν τερματίσει το πρόγραμμα.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

1.3 Επίλυση προβλήματος συγχρονισμού

```
/*
 * kgarten.c
 *
 * A kindergarten simulator.
 * Bad things happen if teachers and children
 * are not synchronized properly.
 *
 *
 * Author:
 * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
 *
 * Additional Authors:
 * Stefanos Gerangelos <sgerag@cslab.ece.ntua.gr>
 * Anastassios Nanos <ananos@cslab.ece.ntua.gr>
 * Operating Systems course, ECE, NTUA
 *
 */


#include <time.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
#include <unistd.h>
#include <pthread.h>
#include <semaphore.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)

/* A virtual kindergarten */
struct kgarten_struct {

	/*
	 * Here you may define any mutexes / condition variables / other variables
	 * you may need.
	 */
   pthread_cond_t cond;
	/*
* You may NOT modify anything in the structure below this
	 * point.
```

```
	 */
	int vt;
	int vc;
	int ratio;

	pthread_mutex_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 */

	struct kgarten_struct *kg;
	int is_child;  /* Nonzero if this thread simulates children, zero otherwise */

	int thrid;     /* Application-defined thread id */
	int thrcnt;
	unsigned int rseed;
};
```

Ομάδα A05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
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 *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_count child_threads c_t_ratio\n\n"
        "Exactly two argument required:\n"
        "    thread_count: Total number of threads to create.\n"
        "    child_threads: The number of threads simulating children.\n"
        "    c_t_ratio: The allowed ratio of children to teachers.\n\n",
        argv0);
    exit(1);
}

void bad_thing(int thrid, int children, int teachers)
{
    int thing, sex;
    int namecnt, nameidx;
    char *name, *p;
    char buf[1024];

    char *things[] = {
        "Little %s put %s finger in the wall outlet and got electrocuted!",
        "Little %s fell off the slide and broke %s head!",
```

```
        "Little %s was playing with matches and lit %s hair on fire!",
        "Little %s drank a bottle of acid with %s lunch!",
        "Little %s caught %s hand in the paper shredder!",
        "Little %s wrestled with a stray dog and it bit %s finger off!"
};
char *boys[] = {
        "George", "John", "Nick", "Jim", "Constantine",
        "Chris", "Peter", "Paul", "Steve", "Billy", "Mike",
        "Vangelis", "Antony"
};
char *girls[] = {
        "Maria", "Irene", "Christina", "Helena", "Georgia", "Olga",
        "Sophie", "Joanna", "Zoe", "Catherine", "Marina", "Stella",
        "Vicky", "Jenny"
};

thing = rand() % 4;
sex = rand() % 2;

namecnt = sex ? sizeof(boys)/sizeof(boys[0]) : sizeof(girls)/sizeof(girls[0]);
nameidx = rand() % namecnt;
name = sex ? boys[nameidx] : girls[nameidx];
```

```
	p = buf;
	p += sprintf(p, "*** Thread %d: Oh no! ", thrid);
	p += sprintf(p, things[thing], name, sex ? "his" : "her");
	p += sprintf(p, "\n*** Why were there only %d teachers for %d children?!\n",
		teachers, children);

	/* Output everything in a single atomic call */
	printf("%s", buf);
}

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)/(thr->kg->ratio))>=(thr->kg->vt))||((thr->kg->vt)==0))
	//#childs / ratio >= #teachers OR #teachers == 0
		pthread_cond_wait(&thr->kg->cond,&thr->kg->mutex);
```

```
   //atomically unlocks the mutex and waits for the condition variable cond to be
signalled.
      fprintf(stderr, "THREAD %d: CHILD ENTER\n", thr->thrid);

      ++(thr->kg->vc);
      pthread_mutex_unlock(&thr->kg->mutex);
}

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);
      }

   pthread_mutex_lock(&thr->kg->mutex);

      fprintf(stderr, "THREAD %d: CHILD EXIT\n", thr->thrid);
      //restarts all the threads that are waiting on the condition variable cond.
      --(thr->kg->vc);
      pthread_cond_broadcast(&thr->kg->cond);
```

```
	//--(thr->kg->vc);

	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);
	}

	pthread_mutex_lock(&thr->kg->mutex);

	fprintf(stderr, "THREAD %d: TEACHER ENTER\n", thr->thrid);
   //restarts all the threads that are waiting on the condition variable cond.
	++(thr->kg->vt);
	pthread_cond_broadcast(&thr->kg->cond);
	//++(thr->kg->vt);

	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->vt)-1)*(thr->kg->ratio))<=(thr->kg->vc))
	//(#teachers-1) * ratio <= #childs
		pthread_cond_wait(&thr->kg->cond,&thr->kg->mutex);
   // atomically unlocks the mutex and waits for the condition variable cond to be
signalled.
   fprintf(stderr, "THREAD %d: TEACHER EXIT\n", thr->thrid);
	--(thr->kg->vt);


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

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
/*
 * Verify the state of the kindergarten.
 */
void verify(struct thread_info_struct *thr)
{
	struct kgarten_struct *kg = thr->kg;
	int t, c, r;

	c = kg->vc;
	t = kg->vt;
	r = kg->ratio;

	fprintf(stderr, "		Thread %d: Teachers: %d, Children: %d\n",
		thr->thrid, t, c);

	if (c > t * r) {
		bad_thing(thr->thrid, c, t);
		exit(1);
	}
}
```

Ομάδα A05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
/*
 * A single thread.
 * It simulates either a teacher, or a child.
 */
void *thread_start_fn(void *arg)
{
	/* We know arg points to an instance of thread_info_struct */
	struct thread_info_struct *thr = arg;
	char *nstr;

	fprintf(stderr, "Thread %d of %d. START.\n", thr->thrid, thr->thrcnt);

	nstr = thr->is_child ? "Child" : "Teacher";
	for (;;) {
		fprintf(stderr, "Thread %d [%s]: Entering.\n", thr->thrid, nstr);
		if (thr->is_child)
			child_enter(thr);
		else
			teacher_enter(thr);

		fprintf(stderr, "Thread %d [%s]: Entered.\n", thr->thrid, nstr);

		/*
```

```
         * We're inside the critical section,
         * just sleep for a while.
         */
        /* usleep(rand_r(&thr->rseed) % 1000000 / (thr->is_child ? 10000 : 1));
*/
        pthread_mutex_lock(&thr->kg->mutex);
        verify(thr);
        pthread_mutex_unlock(&thr->kg->mutex);

        usleep(rand_r(&thr->rseed) % 1000000);

        fprintf(stderr, "Thread %d [%s]: Exiting.\n", thr->thrid, nstr);
        /* CRITICAL SECTION END */

        if (thr->is_child)
            child_exit(thr);
        else
            teacher_exit(thr);

        fprintf(stderr, "Thread %d [%s]: Exited.\n", thr->thrid, nstr);

        /* Sleep for a while before re-entering */
        /* usleep(rand_r(&thr->rseed) % 100000 * (thr->is_child ? 100 : 1)); */
```

```
            usleep(rand_r(&thr->rseed) % 100000);
            pthread_mutex_lock(&thr->kg->mutex);
            verify(thr);
            pthread_mutex_unlock(&thr->kg->mutex);
        }

        fprintf(stderr, "Thread %d of %d. END.\n", thr->thrid, thr->thrcnt);

        return NULL;
}


int main(int argc, char *argv[])
{
        int i, ret, thrcnt, chldcnt, ratio;
        struct thread_info_struct *thr;
        struct kgarten_struct *kg;

        /*
         * Parse the command line
         */
        if (argc != 4)
                usage(argv[0]);
```

```
if (safe_atoi(argv[1], &thrcnt) < 0 || thrcnt <= 0) {
	fprintf(stderr, "`%s' is not valid for `thread_count'\n", argv[1]);
	exit(1);
}
if (safe_atoi(argv[2], &chldcnt) < 0 || chldcnt < 0 || chldcnt > thrcnt) {
	fprintf(stderr, "`%s' is not valid for `child_threads'\n", argv[2]);
	exit(1);
}
if (safe_atoi(argv[3], &ratio) < 0 || ratio < 1) {
	fprintf(stderr, "`%s' is not valid for `c_t_ratio'\n", argv[3]);
	exit(1);
}



/*
 * Initialize kindergarten and random number generator
 */
srand(time(NULL));

kg = safe_malloc(sizeof(*kg));
   kg->vt = kg->vc = 0;
kg->ratio = ratio;
```

```
    ret = pthread_mutex_init(&kg->mutex, NULL);
    if (ret) {
        perror_pthread(ret, "pthread_mutex_init");
        exit(1);
    }

    ret = pthread_cond_init(&kg->cond, NULL);
    if (ret) {
        perror_pthread(ret, "pthread_cond_init");
        exit(1);
}
    /*
     * Create threads
     */
    thr = safe_malloc(thrcnt * sizeof(*thr));

    for (i = 0; i < thrcnt; i++) {
        /* Initialize per-thread structure */
        thr[i].kg = kg;
        thr[i].thrid = i;
        thr[i].thrcnt = thrcnt;
        thr[i].is_child = (i < chldcnt);
        thr[i].rseed = rand();
```

```
        /* Spawn new thread */
        ret = pthread_create(&thr[i].tid, NULL, thread_start_fn, &thr[i]);
        if (ret) {
                perror_pthread(ret, "pthread_create");
                exit(1);
        }
    }

    /*
     * Wait for all threads to terminate
     */
    for (i = 0; i < thrcnt; i++) {
        ret = pthread_join(thr[i].tid, NULL);
        if (ret) {
                perror_pthread(ret, "pthread_join");
                exit(1);
                                }
    }

    printf("OK.\n");
      return 0;
}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του εκτελέσιμου (συνεχίζεται επ' αόριστον)

```
THREAD 290: TEACHER ENTER
Thread 290 [Teacher]: Entered.
            Thread 290: Teachers: 183, Children: 92
            Thread 7: Teachers: 183, Children: 92
Thread 7 [Child]: Entering.
THREAD 7: CHILD ENTER
Thread 7 [Child]: Entered.
            Thread 7: Teachers: 183, Children: 93
            Thread 203: Teachers: 183, Children: 93
Thread 203 [Teacher]: Entering.
THREAD 203: TEACHER ENTER
Thread 203 [Teacher]: Entered.
            Thread 203: Teachers: 184, Children: 93
            Thread 188: Teachers: 184, Children: 93
Thread 188 [Teacher]: Entering.
THREAD 188: TEACHER ENTER
Thread 188 [Teacher]: Entered.
            Thread 188: Teachers: 185, Children: 93
Thread 284 [Teacher]: Exiting.
THREAD 284: TEACHER EXIT
Thread 284 [Teacher]: Exited.
Thread 157 [Teacher]: Exiting.
THREAD 157: TEACHER EXIT
Thread 157 [Teacher]: Exited.
Thread 6 [Child]: Exiting.
THREAD 6: CHILD EXIT
Thread 6 [Child]: Exited.
            Thread 65: Teachers: 183, Children: 92
Thread 65 [Child]: Entering.
THREAD 65: CHILD ENTER
Thread 65 [Child]: Entered.
            Thread 65: Teachers: 183, Children: 93
Thread 293 [Teacher]: Exiting.
THREAD 293: TEACHER EXIT
Thread 293 [Teacher]: Exited.
            Thread 153: Teachers: 182, Children: 93
Thread 153 [Teacher]: Entering.
THREAD 153: TEACHER ENTER
Thread 153 [Teacher]: Entered.
            Thread 153: Teachers: 183, Children: 93
Thread 30 [Child]: Exiting.
THREAD 30: CHILD EXIT
Thread 30 [Child]: Exited.
            Thread 242: Teachers: 183, Children: 92
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Απαντήσεις:

1. Όταν ένας δάσκαλος επιχειρεί να φύγει ελέγχει αν το επιτρέπει η αναλογία παιδιών – δασκάλων που είναι ήδη μέσα στο σταθμό . Το ίδιο συμβαίνει και όταν ένα παιδί επιχειρεί να μπει . Τόσο τα παιδιά που φεύγουν όσο και οι δάσκαλοι που μπαίνουν στέλνουν σήμα στους δασκάλους και στα παιδιά που περιμένουν να βγουν και να μπουν αντίστοιχα, να συνεχίσουν. Ο πρώτος που θα πάρει το κλειδί ελέγχει αν το επιτρέπει η συνθήκη του να συνεχίσει αλλιώς περιμένει (wait) και ξεκλειδώνει για να μπορέσει να μπει κάποιος άλλως.

2. Κάθε φορά που περισσότερα από ένα νήματα ζητούν το κλειδί για να έχουν έλεγχο στο κρίσιμο τμήμα , παρατηρούμε καταστάσεις συναγωνισμού στον κώδικα. Επίσης, καταστάσεις συναγωνισμού παρατηρούνται κάθε φορά που περισσότερα από ένα νήματα ζητούν το κλειδί στις συναρτήσεις εισόδου και εξόδου των παιδιών ή των δασκάλων. Για παράδειγμα αν ένα παιδί φύγει από τον παιδικό σταθμό και προσπαθήσει να καλέσει την verify για την επαλήθευση αλλά εκείνη την στιγμή το κλειδί το έχει ένα παιδί που μπαίνει στον σταθμό τότε το αποτέλεσμα της επαλήθευσης για το πρώτο παιδί που βγήκε θα είναι διαφορετικό από το αληθινό, δηλαδή από την πραγματική αναλογία την στιγμή που μόλις έφυγε.