simplesync.c

```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>
#include <stdatomic.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 lock;
#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_add_and_fetch(ip,1);
        } else {
            pthread_mutex_lock(&lock);
            /* You cannot modify the following line */
            ++(*ip);
            pthread_mutex_unlock(&lock);
        }
    }
    fprintf(stderr, "Done increasing variable.\n");
```

```c

    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(&lock);
            /* You cannot modify the following line */
            --(*ip);
            pthread_mutex_unlock(&lock);
        }
    }
    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;
    if(pthread_mutex_init(&lock,NULL) != 0)
            {perror("mutex"); exit(1);}

    /*
     * 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)
```

```
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 }
129
```