

simplesync.c

```

1  /*
2  * simplesync.c
3  *
4  * A simple synchronization exercise.
5  *
6  * Vangelis Koukis <vkoukis@cslab.ece.ntua.gr>
7  * Operating Systems course, ECE, NTUA
8  *
9  */
10
11 #include <errno.h>
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <unistd.h>
15 #include <pthread.h>
16 #include <stdatomic.h>
17
18 /*
19 * POSIX thread functions do not return error numbers in errno,
20 * but in the actual return value of the function call instead.
21 * This macro helps with error reporting in this case.
22 */
23 #define perror_thread(ret, msg) \
24     do { errno = ret; perror(msg); } while (0)
25
26 #define N 10000000
27
28 /* Dots indicate lines where you are free to insert code at will */
29 /* ... */
30 pthread_mutex_t lock;
31 #if defined(SYNC_ATOMIC) ^ defined(SYNC_MUTEX) == 0
32 # error You must #define exactly one of SYNC_ATOMIC or SYNC_MUTEX.
33 #endif
34
35 #if defined(SYNC_ATOMIC)
36 # define USE_ATOMIC_OPS 1
37 #else
38 # define USE_ATOMIC_OPS 0
39 #endif
40
41 void *increase_fn(void *arg)
42 {
43     int i;
44     volatile int *ip = arg;
45
46     fprintf(stderr, "About to increase variable %d times\n", N);
47     for (i = 0; i < N; i++) {
48         if (USE_ATOMIC_OPS) {
49             __sync_add_and_fetch(ip, 1);
50         } else {
51             pthread_mutex_lock(&lock);
52             /* You cannot modify the following line */
53             ++(*ip);
54             pthread_mutex_unlock(&lock);
55         }
56     }
57     fprintf(stderr, "Done increasing variable.\n");

```

```
58
59     return NULL;
60 }
61
62 void *decrease_fn(void *arg)
63 {
64     int i;
65     volatile int *ip = arg;
66
67     fprintf(stderr, "About to decrease variable %d times\n", N);
68     for (i = 0; i < N; i++) {
69         if (USE_ATOMIC_OPS) {
70             __sync_sub_and_fetch(ip, 1);
71         } else {
72             pthread_mutex_lock(&lock);
73             /* You cannot modify the following line */
74             --(*ip);
75             pthread_mutex_unlock(&lock);
76         }
77     }
78     fprintf(stderr, "Done decreasing variable.\n");
79
80     return NULL;
81 }
82
83
84 int main(int argc, char *argv[])
85 {
86     int val, ret, ok;
87     pthread_t t1, t2;
88
89     /*
90      * Initial value
91      */
92     val = 0;
93     if(pthread_mutex_init(&lock, NULL) != 0)
94         {perror("mutex"); exit(1);}
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 }
129
```