```
 1 // threadtest.cc
 2 //  Simple test case for the threads assignment.
 3 //
 4 //  Create two threads, and have them context switch
 5 //  back and forth between themselves by calling Thread::Yield,
 6 //  to illustratethe inner workings of the thread system.
 7 //
 8 // Copyright (c) 1992-1993 The Regents of the University of California.
 9 // All rights reserved.  See copyright.h for copyright notice and limitation
10 // of liability and disclaimer of warranty provisions.
11
12 #include "copyright.h"
13 #include "system.h"
14
15 //----------------------------------------------------------------------
16 // the function template
17 // Inc and Dec execute value++ and value-- for once.
18 //
19 //  "which" is simply a number identifying the thread, for debugging
20 //  purposes.
21 //----------------------------------------------------------------------
22
23 //the shared variable.
24 int value=0;
25
26 //increase value by one
27 void Inc(_int which)
28 {
29     int a=value;
30     a++;
31     value=a;
32     printf("**** Inc thread %d new value %d\n", (int) which, value);
33 }
34
35 //decrease value by one
36 void Dec(_int which)
37 {
38     int a=value;
39     a--;
40     value=a;
41     printf("**** Dec thread %d new value %d\n", (int) which, value);
42 }
43
44 //exercise 1: two Inc threads and two Dec threads, and implement the interleaving
45 //so that value is equal to a given value when all the four threads ends.
46
47
48
49 //After executing TestValueOne(), the value should be one.
50 //1. implement the new version of Inc: Inc_v1
51 void Inc_v1(_int which)
52 {
53     //fill your code
54
55     //same logic as Inc
```

```
 56     int a=value;
 57     a++;
 58     value=a;
 59     printf("**** Inc thread %d new value %d\n", (int) which, value);
 60
 61     //induce context switch
 62     //using Yield()
 63     currentThread -> Yield();
 64 }
 65
 66 //2. implement the new version of Dec: Dec_v1
 67 void Dec_v1(_int which)
 68 {
 69     //fill your code
 70
 71     //same logic as Dec
 72     int a=value;
 73     a--;
 74
 75     //induce context switch
 76     //using Yield()
 77     currentThread -> Yield();
 78
 79     value=a;
 80     printf("**** Dec thread %d new value %d\n", (int) which, value);
 81 }
 82
 83 //3. implement TestValueOne by create two threads with Inc_v1 and two threads
    with Dec_v1
 84 // you should pass the checking at the end, printing "congratulations! passed."
 85 void TestValueOne()
 86 {
 87     value=0;
 88     printf("enter TestValueOne, value=%d...\n", value);
 89
 90     //1. fill your code here.
 91
 92     Thread *t1 = new Thread("Inc_1");
 93     t1 -> Fork(Inc_v1, 1, 0);
 94
 95     Thread *t2 = new Thread("Dec_1");
 96     t2 -> Fork(Dec_v1, 2, 0);
 97
 98     Thread *t3 = new Thread("Inc_2");
 99     t3 -> Fork(Inc_v1, 3, 0);
100
101     Thread *t4 = new Thread("Dec_2");
102     t4 -> Fork(Dec_v1, 4, 1);
103
104     currentThread -> Join(t4);  //current state should wait for all those threads
     to complete
105
106
107
108     //2. checking the value. you should not modify the code or add any code lines
```

```
108  behind
109     //this section.
110     if(value==1)
111         printf("congratulations! passed.\n");
112     else
113         printf("value=%d, failed.\n", value);
114 }
115
116
117
118 //After executing TestValueMinusOne(), the value should be -1.
119 //1. implement the new version of Inc: Inc_v2
120 void Inc_v2(_int which)
121 {
122     //fill your code
123
124     //same logic as Inc
125     int a=value;
126     a++;
127
128     //induce context switch
129     //using Field()
130     currentThread -> Yield();
131
132     value=a;
133     printf("**** Inc thread %d new value %d\n", (int) which, value);
134
135 }
136
137 //2. implement the new version of Dec: Dec_v2
138 void Dec_v2(_int which)
139 {
140     //fill your code
141
142     //same logic as Dec
143     int a=value;
144     a--;
145
146     //induce context switch
147     //using Yield()
148     currentThread -> Yield();
149
150     value=a;
151     printf("**** Dec thread %d new value %d\n", (int) which, value);
152 }
153
154 //3. implement TestValueMinusOne by create two threads with Inc_v2 and two
    threads with Dec_v2
155 // you should pass the checking at the end, printing "congratulations! passed."
156 void TestValueMinusOne()
157 {
158     value=0;
159     printf("enter TestValueMinusOne, value=%d...\n", value);
160
161     //1. fill your code
```

```
162     Thread *t1 = new Thread("Inc_1");
163     t1 -> Fork(Inc_v2, 1, 0);
164 
165     Thread *t2 = new Thread("Dec_1");
166     t2 -> Fork(Dec_v2, 2, 0);
167 
168     Thread *t3 = new Thread("Inc_2");
169     t3 -> Fork(Inc_v2, 3, 0);
170 
171     Thread *t4 = new Thread("Dec_2");
172     t4 -> Fork(Dec_v2, 4, 1);
173 
174     currentThread -> Join(t4);  //current state should wait for all those threads
     to complete
175 
176     //2. checking the value. you should not modify the code or add any code lines
     behind
177     //this section.
178     if(value==-1)
179         printf("congratulations! passed.\n");
180     else
181         printf("value=%d, failed.\n", value);
182 }
183 
184 
185 //Exercise 2: offer an implementation of Inc_Consistent and Dec_Consistent so
    that
186 //no matter what kind of interleaving occurs, the result value should be
    consistent.
187 
188 //1. Declare any paramters here.
189 
190 //fill your code
191 
192 //Lock ver.
193 Lock *lock = new Lock("lock");
194 //Semaphore ver.
195 Semaphore *sem = new Semaphore("sem", 1);   // initial value = 1 -> binary
    semaphore, works like Lock()
196 //Condition Variables ver.
197 Condition *con = new Condition("condition");
198 
199 
200 
201 //2. implement the new version of Inc: Inc_Consistent
202 void Inc_Consistent(_int which)
203 {
204     //fill your code
205 
206     //============ Entry Section ==============
207 
208     //Lock ver.
209     lock -> Acquire();
210 
211     //Semaphore ver.
```

```
212     //sem -> P();   // wait until semaphore value > 0, decrement
213
214     //Condtion Variables ver.
215     //con -> Wait(lock);
216
217     //========================================
218
219
220
221     //same logic as Inc
222     int a=value;
223     a++;
224     value=a;
225
226     //induce context switch
227     //using Yield()
228     currentThread -> Yield();
229     //using Sleep
230     //currentThread -> Sleep();
231
232     printf("**** Dec thread %d new value %d\n", (int) which, value);
233
234     //============ Exit Section ==============
235
236     //Lock ver.
237     lock -> Release();
238
239     //Semaphore ver.
240     //sem -> V();   // increment
241
242     //Condtion Variables ver.
243     //con -> Broadcast(lock);
244
245     //========================================
246
247 }
248
249
250 //3. implement the new version of Dec: Dec_Consistent
251 void Dec_Consistent(_int which)
252 {
253     //fill your code
254
255     //============ Entry Section ==============
256
257     //Lock ver.
258     lock -> Acquire();
259
260     //Semaphore ver.
261     //sem -> P();   // wait until semaphore value > 0, decrement
262
263     //Condtion Variables ver.
264     //con -> Wait(lock);
265
266     //========================================
```

```
267
268     //same logic as Dec
269     int a=value;
270     a--;
271
272     //induce context switch
273     //using Yield()
274     currentThread -> Yield();
275     //using Sleep
276     //currentThread -> Sleep();
277
278     value=a;
279     printf("**** Dec thread %d new value %d\n", (int) which, value);
280
281     //============ Exit Section ==============
282
283     //Lock ver.
284     lock -> Release();
285
286     //Semaphore ver.
287     //sem -> V();   // increment
288
289     //Condtion Variables ver.
290     //con -> Broadcast(lock);
291
292     //=========================================
293 }
294
295 //4. implement TestConsistency() by create two threads with Inc_Consistent and
    two threads with Dec_Consistent
296 // you should pass the checking at the end, printing "congratulations! passed."
297 void TestConsistency()
298 {
299     value=0;
300     printf("enter TestConsistency, value=%d...\n", value);
301
302     //fill your code
303     Thread *t1 = new Thread("Inc_1");
304     t1 -> Fork(Inc_Consistent, 1, 0);
305
306     Thread *t2 = new Thread("Dec_1");
307     t2 -> Fork(Dec_Consistent, 2, 0);
308
309     Thread *t3 = new Thread("Inc_2");
310     t3 -> Fork(Inc_Consistent, 3, 0);
311
312     Thread *t4 = new Thread("Dec_2");
313     t4 -> Fork(Dec_Consistent, 4, 1);
314
315     currentThread -> Join(t4);  //current state should wait for all those threads
     to complete
316
317     //2. checking the value. you should not modify the code or add any code lines
     behind
318     //this section.
```

```
319     if(value==0)
320         printf("congratulations! passed.\n");
321     else
322         printf("value=%d, failed.\n", value);
323 }
324
325
326
327 //======================================================================
328 //                        Demonstrate Your Result
329 //======================================================================
330
331 //select the function that you want to test.
332 void
333 ThreadTest()
334 {
335     int loopTimes=0;
336     DEBUG('t', "Entering SimpleTest");
337     //for exercise 1.
338     //TestValueOne();
339     //TestValueMinusOne();
340
341     //for exercise 2.
342     TestConsistency();
343 }
344
```