```
1   CS 202, Fall 2021
2   Handout 3 (Class 4)
3
4   1. Example to illustrate interleavings: say that thread A executes f()
5   and thread B executes g(). (Here, we are using the term "thread"
6   abstractly. This example applies to any of the approaches that fall
7   under the word "thread".)
8
9       a. [this is pseudocode]
10
11          int x;
12
13          int main(int argc, char** argv) {
14
15              tid tid1 = thread_create(f, NULL);
16              tid tid2 = thread_create(g, NULL);
17
18              thread_join(tid1);
19              thread_join(tid2);
20
21              printf("%d\n", x);
22          }
23
24          void f()
25          {
26              x = 1;
27              thread_exit();
28          }
29
30          void g()
31          {
32              x = 2;
33              thread_exit();
34          }
35
36
37          What are possible values of x after A has executed f() and B has
38          executed g()? In other words, what are possible outputs of the
39          program above?
40
41
42
43      b.  Same question as above, but f() and g() are now defined as
44      follows:
45
46          int y = 12;
47
48          f() { x = y + 1; }
49          g() { y = y * 2; }
50
51          What are the possible values of x?
52
53
54
55      c. Same question as above, but f() and g() are now defined as
56      follows:
57
58          int x = 0;
59          f() { x = x + 1; }
60          g() { x = x + 2; }
61
62          What are the possible values of x?
63
```

```
64  2. Linked list example
65
66      struct List_elem {
67          int data;
68          struct List_elem* next;
69      };
70
71      List_elem* head = 0;
72
73      insert(int data) {
74          List_elem* l = new List_elem;
75          l->data = data;
76          l->next = head;
77          head = l;
78      }
79
80      What happens if two threads execute insert() at once and we get the
81      following interleaving?
82
83      thread 1: l->next = head
84      thread 2: l->next = head
85      thread 2: head = l;
86      thread 1: head = l;
87
```

```
 88  3. Producer/consumer example:
 89
 90      /*
 91      "buffer" stores BUFFER_SIZE items
 92      "count" is number of used slots. a variable that lives in memory
 93      "out" is next empty buffer slot to fill (if any)
 94      "in" is oldest filled slot to consume (if any)
 95      */
 96
 97      void producer (void *ignored) {
 98          for (;;) {
 99              /* next line produces an item and puts it in nextProduced */
100              nextProduced = means_of_production();
101              while (count == BUFFER_SIZE)
102                  ; // do nothing
103              buffer [in] = nextProduced;
104              in = (in + 1) % BUFFER_SIZE;
105              count++;
106          }
107      }
108
109      void consumer (void *ignored) {
110          for (;;) {
111              while (count == 0)
112                  ; // do nothing
113              nextConsumed = buffer[out];
114              out = (out + 1) % BUFFER_SIZE;
115              count--;
116              /* next line abstractly consumes the item */
117              consume_item(nextConsumed);
118          }
119      }
120
121      /*
122        what count++ probably compiles to:
123        reg1 <-- count      # load
124        reg1 <-- reg1 + 1   # increment register
125        count <-- reg1      # store
126
127        what count-- could compile to:
128        reg2 <-- count      # load
129        reg2 <-- reg2 - 1   # decrement register
130        count <-- reg2      # store
131      */
132
133      What happens if we get the following interleaving?
134
135          reg1 <-- count
136          reg1 <-- reg1 + 1
137          reg2 <-- count
138          reg2 <-- reg2 - 1
139          count <-- reg1
140          count <-- reg2
141
```

```
142
143  4. Some other examples. What is the point of these?
144
145      [From S.V. Adve and K. Gharachorloo, IEEE Computer, December 1996,
146      66-76. http://rsim.cs.uiuc.edu/~sadve/Publications/computer96.pdf]
147
148      a. Can both "critical sections" run?
149
150          int flag1 = 0, flag2 = 0;
151
152          int main () {
153              tid id = thread_create (p1, NULL);
154              p2 (); thread_join (id);
155          }
156
157          void p1 (void *ignored) {
158              flag1 = 1;
159              if (!flag2) {
160                  critical_section_1 ();
161              }
162          }
163
164          void p2 (void *ignored) {
165              flag2 = 1;
166              if (!flag1) {
167                  critical_section_2 ();
168              }
169          }
170
171      b. Can use() be called with value 0, if p2 and p1 run concurrently?
172
173          int data = 0, ready = 0;
174
175          void p1 () {
176              data = 2000;
177              ready = 1;
178          }
179          int p2 () {
180              while (!ready) {}
181              use(data);
182          }
183
184      c. Can use() be called with value 0?
185
186          int a = 0, b = 0;
187
188          void p1 (void *ignored) { a = 1; }
189
190          void p2 (void *ignored) {
191            if (a == 1)
192              b = 1;
193          }
194
195          void p3 (void *ignored) {
196            if (b == 1)
197              use (a);
198          }
```