

```

1 Handout for CS 439
2 Class 3
3 22 January 2013
4
5 1. Example to illustrate interleavings: say that thread A executes f()
6 and thread B executes g(). (Here, we are using the term "thread"
7 abstractly. This example applies to any of the approaches that fall
8 under the word "thread".)
9

```

a. [this is pseudocode]

```

10     int x;
11
12     int main(int argc, char** argv) {
13
14         tid tid1 = thread_create(f, NULL);
15         tid tid2 = thread_create(g, NULL);
16
17         thread_join(tid1);
18         thread_join(tid2);
19
20         printf("%d\n", x);
21     }
22
23
24     void f()
25     {
26         x = 1;
27         thread_exit();
28     }
29
30
31     void g()
32     {
33         x = 2;
34         thread_exit();
35     }
36
37

```

What are possible values of x after A has executed f() and B has executed g()? In other words, what are possible outputs of the program above?

b. Same question as above, but f() and g() are now defined as follows:

```

46     int y = 12;
47
48     f() { x = y + 1; }
49     g() { y = y * 2; }
50
51

```

What are the possible values of x?

c. Same question as above, but f() and g() are now defined as follows:

```

58     int x = 0;
59     f() { x = x + 1; }
60     g() { x = x + 2; }
61
62

```

What are the possible values of x?

64

65 2. Linked list example

```

66
67     struct List_elem {
68         int data;
69         struct List_elem* next;
70     };
71
72     List_elem* head = 0;
73
74     insert(int data) {
75         List_elem* l = new List_elem;
76         l->data = data;
77         l->next = head;
78         head = l;
79     }
80

```

81 What happens if two threads execute insert() at once and we get the following interleaving?

```

82
83
84     thread 1: l->next = head
85     thread 2: l->next = head
86     thread 2: head = l;
87     thread 1: head = l;
88

```

Jan 22, 13 16:04

I03-handout.txt

Page 3/4

```

89 3. Producer/consumer example:
90
91  /*
92  "buffer" stores BUFFER_SIZE items
93  "count" is number of used slots. a variable that lives in memory
94  "out" is next empty buffer slot to fill (if any)
95  "in" is oldest filled slot to consume (if any)
96  */
97
98  void producer (void *ignored) {
99      for (;;) {
100         /* next line produces an item and puts it in nextProduced */
101         nextProduced = means_of_production();
102         while (count == BUFFER_SIZE)
103             ; // do nothing
104         buffer [in] = nextProduced;
105         in = (in + 1) % BUFFER_SIZE;
106         count++;
107     }
108 }
109
110 void consumer (void *ignored) {
111     for (;;) {
112         while (count == 0)
113             ; // do nothing
114         nextConsumed = buffer[out];
115         out = (out + 1) % BUFFER_SIZE;
116         count--;
117         /* next line abstractly consumes the item */
118         consume_item(nextConsumed);
119     }
120 }
121
122 /*
123 what count++ probably compiles to:
124 reg1 <-- count      # load
125 reg1 <-- reg1 + 1   # increment register
126 count <-- reg1     # store
127
128 what count-- could compile to:
129 reg2 <-- count      # load
130 reg2 <-- reg2 - 1   # decrement register
131 count <-- reg2     # store
132 */
133
134 What happens if we get the following interleaving?
135
136 reg1 <-- count
137 reg1 <-- reg1 + 1
138 reg2 <-- count
139 reg2 <-- reg2 - 1
140 count <-- reg1
141 count <-- reg2
142

```

Jan 22, 13 16:04

I03-handout.txt

Page 4/4

```

143
144 4. Some other examples. What is the point of these?
145
146 [From S.V. Adve and K. Gharachorloo, IEEE Computer, December 1996,
147 66-76. http://rsim.cs.uiuc.edu/~sadve/Publications/computer96.pdf]
148
149 a. Can both "critical sections" run?
150
151     int flag1 = 0, flag2 = 0;
152
153     int main () {
154         tid id = thread_create (p1, NULL);
155         p2 (); thread_join (id);
156     }
157
158     void p1 (void *ignored) {
159         flag1 = 1;
160         if (!flag2) {
161             critical_section_1 ();
162         }
163     }
164
165     void p2 (void *ignored) {
166         flag2 = 1;
167         if (!flag1) {
168             critical_section_2 ();
169         }
170     }
171
172 b. Can use() be called with value 0, if p2 and p1 run concurrently?
173
174     int data = 0, ready = 0;
175
176     void p1 () {
177         data = 2000;
178         ready = 1;
179     }
180     int p2 () {
181         while (!ready) {}
182         use(data);
183     }
184
185 c. Can use() be called with value 0?
186
187     int a = 0, b = 0;
188
189     void p1 (void *ignored) { a = 1; }
190
191     void p2 (void *ignored) {
192         if (a == 1)
193             b = 1;
194     }
195
196     void p3 (void *ignored) {
197         if (b == 1)
198             use (a);
199     }
200

```