```
CS5600 Week05.b

The handout from the last class gave examples of race conditions.
The following panels demonstrate the use of concurrency primitives
(mutexes, etc.). We are using concurrency primitives to eliminate
race conditions (see items 1 and 2a) and improve scheduling (see item 2b).


1. Protecting the linked list...

    Mutex list_mutex;

    insert(int data) {
      List_elem* l = new List_elem;
      l->data = data;

      acquire(&list_mutex);

      l->next = head;
      head = l;

      release(&list_mutex);
    }

```



```
2. Producer/consumer revisited [also known as bounded buffer]

2a. Producer/consumer [bounded buffer] with mutexes

    Mutex mutex;

    void producer (void *ignored) {
      for (;;) {
        /* next line produces an item and puts it in nextProduced */
        nextProduced = means_of_production();

        acquire(&mutex);
        while (count == BUFFER_SIZE) {
          release(&mutex);
          yield(); /* or schedule() */
          acquire(&mutex);
        }

        buffer [in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
        count++;
        release(&mutex);
      }
    }

    void consumer (void *ignored) {
      for (;;) {

        acquire(&mutex);
        while (count == 0) {
          release(&mutex);
          yield(); /* or schedule() */
          acquire(&mutex);
        }

        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        count--;
        release(&mutex);

        /* next line abstractly consumes the item */
        consume_item(nextConsumed);
      }
    }

```

```
70
71  2b. Producer/consumer [bounded buffer] with mutexes and condition variables
72
73      Mutex mutex;
74      Cond nonempty;
75      Cond nonfull;
76
77      void producer (void *ignored) {
78        for (;;) {
79          /* next line produces an item and puts it in nextProduced */
80          nextProduced = means_of_production();
81
82          acquire(&mutex);
83          while (count == BUFFER_SIZE)
84            cond_wait(&nonfull, &mutex);
85
86          buffer [in] = nextProduced;
87          in = (in + 1) % BUFFER_SIZE;
88          count++;
89          cond_signal(&nonempty, &mutex);
90          release(&mutex);
91        }
92      }
93
94      void consumer (void *ignored) {
95        for (;;) {
96
97          acquire(&mutex);
98          while (count == 0)
99            cond_wait(&nonempty, &mutex);
100
101         nextConsumed = buffer[out];
102         out = (out + 1) % BUFFER_SIZE;
103         count--;
104         cond_signal(&nonfull, &mutex);
105         release(&mutex);
106
107         /* next line abstractly consumes the item */
108         consume_item(nextConsumed);
109       }
110     }
111
112
113 Question: why does cond_wait need to both release the mutex and
114 sleep? Why not:
115
116     while (count == BUFFER_SIZE) {
117       release(&mutex);
118       cond_wait(&nonfull);
119       acquire(&mutex);
120     }
121
```