Producer Consumer Problem

This algorithm uses eventcounters and sequencers to solve the producer/consumer (or bounded-buffer) problem.

```
 1   var nextp, nextc: item;
 2        IN, OUT: eventcounter;
 3        T: sequencer;
 4   procedure producer;
 5   begin
 6        var t: integer;
 7        while true do begin
 8             (* produce item in nextp *)
 9             t := ticket(T);
10             await(IN, t);
11             await(OUT, t - N + 1);
12             buffer[(t + 1) mod N] := nextp;
13             advance(IN);
14        end;
15   end;
16   procedure consumer;
17   begin
18        var i: integer;
19        i := 1;
20        while true do begin
21             await(IN, i);
22             nextc := buffer[i mod N];
23             (* consume item in nextc *)
24             advance(OUT);
25             i := i + 1;
26        end;
27   end;
28   begin
29        parbegin
30             consumer;
31             producer;
32        parend
33   end.
```

lines 1-3 Here, *nextp* is the item the consumer produces, and *nextc* the item that the consumer consumes. The eventcounter *IN* synchronizes the producers and consumers so that at most one at a time accesses the buffer. *OUT*

lines 4-15 This procedure sequences the producers so that only one at a time is writing to the buffer.

lines 9-10 The variable t is the sequence number. The next sequence number is generated, and the process blocks until the eventcounter *IN* reaches that value.

line 11 This blocks until the appropriate element in the buffer is available. Then it proceeds.

line 13 By incrementing the eventcounter, this allows the next producer in to add *nextp* to element $(t + 1) \bmod N$. It also allows the next consumer consume the item in element $t \bmod N$.

lines 16-27 This procedure sequences the producers so that only one at a time is writing to the buffer.

line 21 This blocks the consumer until a producer puts an item in element i. Note this variable is the same as in the producer, tying the two of them together.

line 24 The consumer has removed one more item, so the consumer increments the eventcounter *OUT*. Again, the producer waiting on this element being available for storage is signaled.

lines 29-32 This starts two concurrent processes, the *consumer* and the *producer*.