Producer Consumer Problem

This algorithm uses blocking send and receive primitives to solve the producer/consumer (or bounded-buffer) problem. In this solution, the buffer size depends on the capacity of the link.

```
 1 var nextp, nextc: item;
 2 procedure producer;
 3 begin
 4            while true do begin
 5                        (* produce item in nextp *)
 6                        send("Consumerprocess", nextp);
 7            end;
 8 end;
 9 procedure consumer;
10 begin
11            while true do begin
12                        receive("Producerprocess", nextc);
13                        (* consume item in nextc *)
14            end;
15 end;
16 begin
17            parbegin
18                        Consumerprocess: consumer;
19                        Producerprocess: producer;
20            parend
21 end.
```

line 1 Here, *nextp* is the item the consumer produces, and *nextc* the item that the consumer consumes.

lines 2-8 This procedure simply generates items and sends them to the consumer process (named *Consumerprocess*). Suppose the capacity of the link is *n* items. If *n* items are waiting to be consumed, and the producer tries to *send* the *n*+1-st item, the producer will block (suspend) until the consumer has removed one item from the link (i.e., done a *receive* on the producer process). Note the name of the consumer process is given explicitly, so this is an example of "explicit naming" or "direct communication." Also, since the *send* is blocking, it is an example of "synchronous communication."

lines 9-15 This code simply receives items from the producer process (named *Producerprocess*) and consumes them. If when the receive statement is executed there are no items in the link, the consumer will block (suspend) until the producer has put an item from the link (i.e., done a *send* to the consumer process). Note the name of the producer process is given explicitly; again this is an example of "explicit naming" or "direct communication." Also, since the *receive* is blocking, it is an example of "synchronous communication."

lines 17-20 This starts two concurrent processes, the *Consumerprocess* and the *Producerprocess*.