Producer/Consumer Problem

Introduction

This algorithm uses semaphores to solve the producer/consumer (or bounded buffer) problem.

Algorithm

```
 1  var        buffer: array [0..n-1] of item;
 2             full, empty, mutex: semaphore;
 3             nextp, nextc: item;
 4  begin
 5      full := 0;
 6      empty := n;
 7      mutex := 1;
 8      parbegin
 9          repeat                          (* producer process *)
10              (* produce an item in nextp *)
11              down(empty);
12              down(mutex);
13              (* deposit nextp in buffer *)
14              up(mutex);
15              up(full);
16          until false;
17          repeat                          (* consumer process *)
18              down(full);
19              down(mutex);
20              (* extract an item in nextc *)
21              up(mutex);
22              up(empty);
23              (* consume the item in nextc *)
24          until false;
25      parend ;
26  end .
```

Comments

lines 1-3	Here, `buffer` is the shared buffer, and contains `n` spaces; `full` is a semaphore the value of which is the number of filled slots in the buffer, `empty` is a semaphore the value of which is the number of emoty slots in the buffer, and `mutex` is a semaphore used to enforce mutual exclusion (so only one process can access the buffer at a time). `nextp` and `nextc` are the items produced by the producer and consumed by the consumer, respectively.
line 5-7	This just initializes all the semaphores. It is the only time anything other than a *down* or an *up* operation may be done to them.
line 10	Since the buffer is not accessed while the item is produced, we don't need to put semaphores around this part.
lines 11-13	Depositing an item into the buffer, however, does require that the producer process obtain exclusive access to the buffer. First, the producer checks that there is an empty slot in the buffer for the new item and, if not, waits until there is (`down(empty)`). When there is, it waits until it can obtain exclusive access to the buffer (`down(mutex)`). Once both these conditions are met, it can safely deposit the item.
lines 14-15	As the producer is done with the buffer, it signals that any other process needing to access the buffer may do so (`up(mutex)`). It then indicates it has put another item into the buffer (`up(full)`).
lines 18-20	Extracting an item from the buffer, however, does require that the consumer process obtain exclu-

	sive access to the buffer. First, the consumer checks that there is a slot in the buffer with an item deposited and, if not, waits until there is (`down(full)`). When there is, it waits until it can obtain exclusive access to the buffer (`down(mutex)`). Once both these conditions are met, it can safely extract the item.
lines 21-22	As the consumer is done with the buffer, it signals that any other process needing to access the buffer may do so (`up(mutex)`). It then indicates it has extracted another item into the buffer (`up(empty)`).
line 23	Since the buffer is not accessed while the item is consumed, we don't need to put semaphores around this part.