Solutions for CP Exercises, September 22

1. Solution for Andrews Ex. 4.14

In order to achieve concurrent deposit and fech operations, we may try to do only the slot allocation under mutual exclusion. Then, however, the filling (and emptying) of the slots can occur out of order. We may choose to loosen the ordering and protect each slot by its private full/empty semaphores. A more conservative option, shown here, is to keep the ordering by passing a signal down the slots indicating that the previous slot has been filled/emptied. This is done through two arrays of semaphores *prev_full* and *prev_empty*:

```
var buf[0..n-1] : T;
    front, rear : integer := 0;
    full : semaphore := 0;
    empty : semaphore := n;
    prev_full[0..n-1] : semaphore := 0;
    prev_empty[0..n-1] : semaphore := 0;
    mutex_P, mutex_C : semaphore := 1;

    V(prev_full);  V(pref_empty);              — first slot OK
```

```
process Producer[i : 1..M] =
  var data : T;
      inpos : integer;
  repeat
    data := produce;
    P(empty);
    P(mutex_P);
    inpos := rear;
    rear := (rear + 1) mod n;
    V(mutex_P);
    buf[inpos] := data;
    P(prev_full[inpos]);
    V(full);
    V(prev_full[(inpos + 1) mod n]);
  forever
```

```
process Consumer[j : 1..N] =
  var result : T;
      outpos : integer;
  repeat
    P(full);
    P(mutex_C);
    outpos := front;
    front := (front + 1) mod n;
    V(mutex_C);
    result := buf[outpos];
    P(prev_empty[outpos]);
    V(empty);
    V(prev_empty[(outpos + 1) mod n]);
    consume result;
  forever
```

This higher degree of concurrency may be beneficial if the data type T is a large datatype such that buffer insertions/removals take significant time.