Distributed implementation of buffers

```
--  This chapter implements procedures safe.multiplex and
--  safe.demultiplex for which the process
--
--    CHAN OF TAGGED.MESSAGE link :
--    CHAN OF INT back :
--    PAR
--      safe.multiplex(from.producer, link, back)
--      safe.demultiplex(link, back, to.consumer)
--
--  behaves like
--
--    PAR i = 0 FOR width
--      buffer(from.producer[i], to.consumer[i])
--

--
--  sizing constants and protocols
--

VAL INT width IS ... :              -- number of virtual channels

VAL INT longest.message IS ... :    -- length of the longest string

PROTOCOL MESSAGE IS INT::[]BYTE :

PROTOCOL SIGNAL
  CASE
    signal
:

PROTOCOL TAGGED.MESSAGE IS INT; INT::[]BYTE :
```

```
--
--  templates for the 'user' procedures
--

PROC producer(VAL INT i, CHAN OF MESSAGE output)
  --  writes some messages to output, using
  --
  --    output ! n :: v
  --
  ... omitted code
:

PROC consumer(VAL INT i, CHAN OF MESSAGE input)
  --  reads some messages from input, using
  --
  --    input ? n :: s
  --
  ... omitted code
:

--
--  component multiplexers and demultiplexers
--

PROC multiplex([]CHAN OF MESSAGE local,
               CHAN OF TAGGED.MESSAGE link )
  WHILE TRUE
    SEQ favoured = 0 FOR width
      INT count :
      [longest.message]BYTE string :
      PRI ALT
        local[favoured] ? count::string
          link ! favoured; count::string
        ALT source = 0 FOR width
          local[source] ? count::string
            link ! source; count::string
:

PROC demultiplex(CHAN OF TAGGED.MESSAGE link,
                 []CHAN OF MESSAGE local     )
  WHILE TRUE
    INT destination, count :
    [longest.message]BYTE string :
    SEQ
      link ? destination; count::string
      local[destination] ! count::string
:
```

```
PROC request.buffer(CHAN OF SIGNAL request,
                    CHAN OF MESSAGE source, sink )
  WHILE TRUE
    INT count :
    [longest.message]BYTE string :
    SEQ
      request ! signal
      source ? count::string
      sink ! count::string
:

--
--  safe.multiplex and safe.demultiplex
--

PROC safe.demultiplex(CHAN OF TAGGED.MESSAGE link,
                      CHAN OF INT back,
                      []CHAN OF MESSAGE local     )
  [width]CHAN OF SIGNAL ack :
  [width]CHAN OF MESSAGE fwd :
  PAR
    demultiplex(link, fwd)
    PAR i = 0 FOR width
      request.buffer(ack[i], fwd[i], local[i])
    WHILE TRUE
      SEQ favoured = 0 FOR width
        PRI ALT
          ack[favoured] ? CASE signal
            back ! favoured
          ALT i = 0 FOR width
            ack[i] ? CASE signal
              back ! i
:

PROC safe.multiplex([]CHAN OF MESSAGE local,
                    CHAN OF TAGGED.MESSAGE link,
                    CHAN OF INT back            )
  [width]CHAN OF SIGNAL ack :
  [width]CHAN OF MESSAGE fwd :
  PAR
    PAR i = 0 FOR width
      request.buffer(ack[i], local[i], fwd[i])
    multiplex(fwd, link)
    WHILE TRUE
      INT i :
      SEQ
        back ? i
        ack[i] ? CASE signal
:
```

```
--
--  program using the mechanism
--

CHAN OF TAGGED.MESSAGE link :
CHAN OF INT back :
PAR
  --
  --  producer processes
  --
  [width]CHAN OF MESSAGE local :
  PAR
    PAR i = 0 FOR width
      producer(i, local[i])
    safe.multiplex(local, link, back)
  --
  --  consumer processes
  --
  [width]CHAN OF MESSAGE local :
  PAR
    safe.demultiplex(link, back, local)
    PAR i = 0 FOR width
      consumer(i, local[i])
```