Semaphore Ring Buffer

Shared Variables

```
var nrfull: semaphore := 0;
    nrempty: semaphore := N;
    mutexP: semaphore := 1;
    mutexC: semaphore := 1;
    buffer: array[0..N-1] of message;
    in, out: 0..N-1:= 0, 0;
```

Producer i

```
loop
    ...
    Create a new message m;
    —One producer at a time
    P(mutexP);
    —Await an empty cell
    P(nrempty);
    buffer[in] := m;
    in := (in + 1) mod N;
    —Signal a full buffer
    V(nrfull);
    —Allow other producers
    V(mutexP);
    ...
    ...
endloop
```

Consumer j

```
loop
    ...
    ...
    —One consumer at a time
    P(mutexC);
    —Await a message
    P(nrfull);
    m := buffer[out];
    out := (out + 1) mod N;
    —Signal an empty buffer
    V(nrempty);
    —Allow other consumers
    V(mutexC);
    Consume message m;
    ...
endloop
```

Weak Reader Preference Solution to the Reader-Writer Problem Using Semaphores

Shared Variables

```
var wmutex, rmutex: semaphore := 1, 1;
  nreaders: integer := 0;
```

A Reader

```
loop
  —Readers enter one at a time
  P(rmutex);
  —First reader waits for reader's turn,
  —then inhibits other writers
  if nreaders = 0 then
    P(wmutex)
  endif;
  nreaders := nreaders + 1;
  —Allow other reader entries/exits
  V(rmutex);
  Perform read operations;
  —Readers exit one at a time
  P(rmutex);
  nreaders := nreaders − 1;
  —Last reader allows writers
  if nreaders = 0 then
    V(wmutex)
  endif;
  —Allow reader entry/exit
  V(rmutex)
endloop
```

A Writer

```
loop
—Each writer operates alone
  P(wmutex);
    Perform write operations;
  V(wmutex)
endloop
```

The Producer/Consumer Problem Using Eventcounts and Sequencers.

Shared Variables

```
var Pticket, Cticket: sequencer;
    In, Out: eventcount;
    buffer: array[0..N-1] of message;
```

Producer i

```
—A variable, t, local
—to each producer
var t: integer;

loop
    . . .
    Create a new message m;
    —One producer at a time
    t := ticket(Pticket);
    await(In, t);
    —Await an empty cell
    await(Out, t-N + 1);
    buffer[t mod N] := m;
    —Signal a full buffer and
    —allow other producers
    advance(In);
    . . .
    . . .
endloop
```

Consumer j

```
—A variable, u, local
—to each consumer
var u: integer;

loop
    . . .
    . . .
    —One consumer at a time
    u := ticket(Cticket);
    await(Out, u);
    —Await a message
    await(In, u + 1);
    m := buffer[u mod N];
    —Signal an empty buffer and
    —allow other consumers
    advance(Out);
    Consume message m;
    . . .
endloop
```

A Weak Reader Preference Solution Using Sequencers and Eventcounts.

Shared Variables

```
var Wticket, Rticket: sequencer;
  Win, Rin: eventcount;
  nreaders: integer := 0;
```

A Reader

```
loop
  ...
  —Readers enter one at a time
  await(Rin, ticket(Rticket));
  —First reader waits for reader's turn,
  —then inhibits other writers
  if nreaders = 0 then
    await(Win, ticket(Wticket))
  endif;
  nreaders := nreaders + 1;
  —Allow other reader entries
  advance(Rin);
  Perform read operations;
  —Readers exit one at a time
  await(Rin, ticket(Rticket));
  nreaders := nreaders − 1;
  —Last reader allows writers
  if nreaders = 0 then
    advance(Win)
  endif;
  —Allow reader entry/exit
  advance(Rin);
  ...
endloop
```

A Writer

```
loop
  ...
  —Each writer operates alone
  await(Win, ticket(Wticket));
  Perform write operations;
  —Allow other writers (or
  —a reader) to lock out
  advance(Win);
  ...
endloop
```

```
01 CIRCULARBUFFER: PROCEDURE OPTIONS (CONCURRENT);
02
03    CIRCULARBUFFERMONITOR: MONITOR;
04       DECLARE (BUFFERS (100)) CHARACTER (80) VARYING;
05       DECLARE (FIRSTBUFFER, LASTBUFFER) FIXED;
06       DECLARE (TOTALBUFFERS, FULLBUFFERS) FIXED;
07       DECLARE (ABUFFERISEMPTY) CONDITION;
08       DECLARE (ABUFFERISFULL) CONDITION;
09
10       DO;
11          FIRSTBUFFER = 1;
12          LASTBUFFER = 1;
13          TOTALBUFFERS = 100;
14          FULLBUFFERS = 0;
15       END;
16
17    SPOOLER: ENTRY (IMAGE);
18       DECLARE (IMAGE) CHARACTER (*) VARYING;
19       IF FULLBUFFERS = TOTALBUFFERS THEN
20          WAIT (ABUFFERISEMPTY);
21       BUFFERS (LASTBUFFER) = IMAGE;
22       LASTBUFFER = MOD (LASTBUFFER, TOTALBUFFERS) + 1;
23       FULLBUFFERS = FULLBUFFERS + 1;
24       SIGNAL(ABUFFERISFULL);
25    END;
26
27    DESPOOLER: ENTRY (IMAGE);
28       DECLARE (IMAGE) CHARACTER (*) VARYING;
29       IF FULLBUFFERS = 0 THEN
30          WAIT (ABUFFERISFULL);
31       IMAGE = BUFFERS (FIRSTBUFFER);
32       FIRSTBUFFER = MOD(FIRSTBUFFER, TOTALBUFFERS) + 1;
33       FULLBUFFERS = FULLBUFFERS - 1;
34       SIGNAL (ABUFFERISEMPTY);
35    END;
36
37 END;
38
39    READCARDS: PROCESS;
40       DECLARE (CARDIMAGE) CHARACTER (80) VARYING;
41       CARDIMAGE = 'MORECARDS';
42       DO WHILE (CARDIMAGE <> 'ENDOFFILE');
43          GET SKIP EDIT (CARDIMAGE) (A(80));
44          CALL SPOOLER (CARDIMAGE);
45       END;
46    END;
47
48    PRINTLINES: PROCESS;
49       DECLARE (LINEIMAGE) CHARACTER (80) VARYING;
50       LINEIMAGE = 'MORECARDS';
51       DO WHILE (LINEIMAGE <> 'ENDOFFILE');
52          CALL DESPOOLER (LINEIMAGE);
53          PUT SKIP EDIT (LINEIMAGE) (A(80));
54       END;
55    END;
56
57 END;
```

CSP/k program for managing a circular buffer.

A Strong Reader Preference Solution Based on Monitors [Hoare, 1978. Copyright © 1978 by Association for Computing Machinery. Reprinted by permission].

```
readers_and_writers: monitor
  var readercount: integer;
    busy: boolean;
    OKtoread, OKtowrite: condition;

  procedure startread;
    begin
      if busy then OKtoread.wait endif;
      readercount := readercount + 1;
      if OKtoread.queue then OKtoread.signal endif
    end startread;

  procedure endread;
    begin
      readercount := readercount - 1;
      if readercount = 0 then OKtowrite.signal endif
    end endread;

  procedure startwrite;
    begin
      if readercount ≠ 0 or busy then OKtowrite.wait endif;
      busy := true
    end startwrite;

  procedure endwrite;
    begin
      busy := false;
      if OKtoread.queue then OKtoread.signal
      else OKtowrite.signal endif
    end endwrite;

begin
    readercount := 0; busy := false
end readers_and_writers;
```