Producer/Consumer Problem

- One or more producers are generating data and placing these in a buffer
- A single consumer is taking items out of the buffer one at time
- Only one producer or consumer may access the buffer at any one time

Producer

```plaintext
producer:
while (true) {
 /* produce item v */
 b[in] = v;
 in++;
}
```
Consumer

consumer:
while (true) {
    while (in <= out)
        /*do nothing */;
    w = b[out];
    out++;
    /* consume item w */
}

Producer/Consumer Problem

![Diagram of Producer/Consumer Problem]

Note: shaded area indicates portion of buffer that is occupied

Figure 5.8 Infinite Buffer for the Producer/Consumer Problem
Producer with Circular Buffer

producer:
while (true) {
    /* produce item v */
    while ((in + 1) % n == out)
        /* do nothing */;
    b[in] = v;
    in = (in + 1) % n
}

Consumer with Circular Buffer

consumer:
while (true) {
    while (in == out)
        /* do nothing */;
    w = b[out];
    out = (out + 1) % n;
    /* consume item w */
}
Figure 5.12 Finite Circular Buffer for the Producer/Consumer Problem

Figure 5.9 An Incorrect Solution to the Infinite Buffer Producer/Consumer Problem Using Binary Semaphores
Table 5.4 Possible Scenario for the Program of Figure 5.9

<table>
<thead>
<tr>
<th>Producer</th>
<th>Consumer</th>
<th>n</th>
<th>n</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

/* program producer_consumer */
int n;
binary_semaphore s = 1;
binary_semaphore delay = 0;
void producer()
{
    while (true)
    {
        produce();
        semWaitB(s);
        append();
        n++;
        if (n==1) semSignalB(delay);
        semSignalB(s);
    }
}
void consumer()
{
    int m; /* a local variable */
    semWaitB(delay);
    while (true)
    {
        semWaitB(s);
        take();
        m--;
        s = m;
        semWaitB(s);
        consume();
        if (m==0) semWaitB(delay);
    }
}
void main()
{
    n = 0;
paxbegin (producer, consumer);
}
/* program producerconsumer */
semaphore n = 0;
semaphore m = 0;
void producer()
{
    while (true)
    {
        produce();
        sendWait(n);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (true)
    {
        semWait(n);
        semWait(m);
        take();
        semSignal(s);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}

Figure 5.11 A Solution to the Infinite-Buffer Producer/Consumer Problem Using Semaphores

/* program boundedbuffer */
const int sizeofbuffer = /* buffer size */;
semaphore s = 1;
semaphore m = 0;
semaphore e = sizeofbuffer;
void producer()
{
    while (true)
    {
        produce();
        sendWait(m);
        semWait(s);
        append();
        semSignal(e);
        semSignal(n)
    }
}
void consumer()
{
    while (true)
    {
        semWait(n);
        semWait(e);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
void main()
{
    parbegin (producer, consumer);
}

Figure 5.13 A Solution to the Bounded-Buffer Producer/Consumer Problem Using Semaphores