# 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

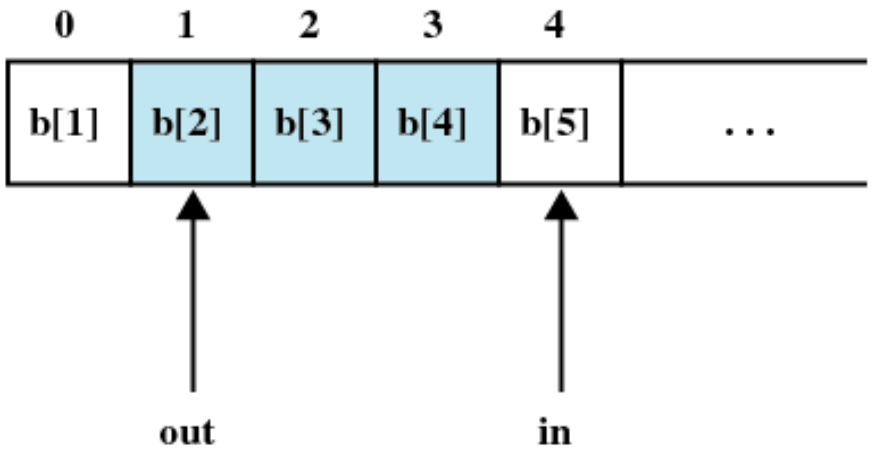
# Producer

```
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




Note: shaded area indicates portion of buffer that is occupied

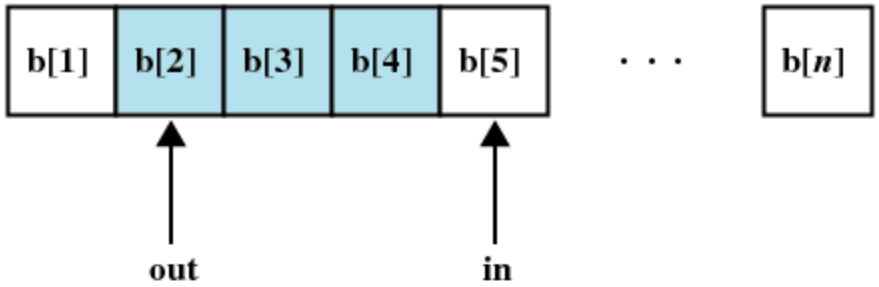
**Figure 5.8 Infinite Buffer for the Producer/Consumer Problem**

# Producer with Circular Buffer

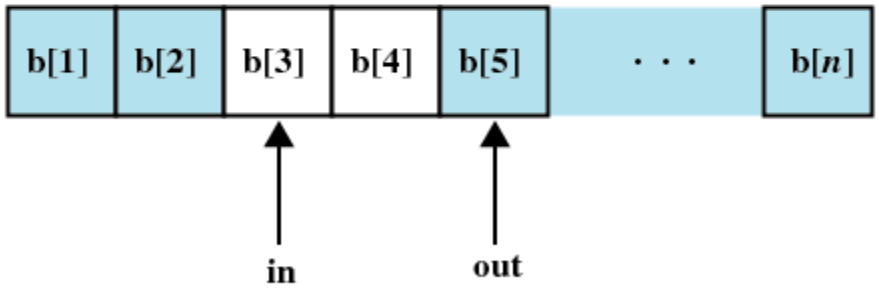
```
producer:
while (true) {
  /* produce item v */
  while ((in + 1) % n == out)
     /* do nothing */;
  b[in] = v;
  in = (in + 1) % n
}
```

# Consumer with Circular Buffer

```
consumer:
while (true) {
  while (in == out)
     /* do nothing */;
  w = b[out];
  out = (out + 1) % n;
  /* consume item w */
}
```




(a)




(b)

**Figure 5.12 Finite Circular Buffer for the Producer/Consumer Problem**

```
/* program producerconsumer */
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()
{
     semWaitB(delay);
     while (true)
     {
          semWaitB(s);
          take();
          n--;
          semSignalB(s);
          consume();
          if (n==0)
               semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```

**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**

| | Producer | Consumer | s | n | Delay |
|---|---|---|---|---|---|
| 1 | | | 1 | 0 | 0 |
| 2 | `semWaitB(s)` | | 0 | 0 | 0 |
| 3 | `n++` | | 0 | 1 | 0 |
| 4 | `if (n==1) (semSignalB(delay))` | | 0 | 1 | 1 |
| 5 | `semSignalB(s)` | | 1 | 1 | 1 |
| 6 | | `semWaitB(delay)` | 1 | 1 | 0 |
| 7 | | `semWaitB(s)` | 0 | 1 | 0 |
| 8 | | `n--` | 0 | 0 | 0 |
| 9 | | `semSignalB(s)` | 1 | 0 | 0 |
| 10 | `semWaitB(s)` | | 0 | 0 | 0 |
| 11 | `n++` | | 0 | 1 | 0 |
| 12 | `if (n==1) (semSignalB(delay))` | | 0 | 1 | 1 |
| 13 | `semSignalB(s)` | | 1 | 1 | 1 |
| 14 | | `if (n==0) (semWaitB(delay))` | 1 | 1 | 1 |
| 15 | | `semWaitB(s)` | 0 | 1 | 1 |
| 16 | | `n--` | 0 | 0 | 1 |
| 17 | | `semSignalB(s)` | 1 | 0 | 1 |
| 18 | | `if (n==0) (semWaitB(delay))` | 1 | 0 | 0 |
| 19 | | `semWaitB(s)` | 0 | 0 | 0 |
| 20 | | `n--` | 0 | **–1** | 0 |
| 21 | | `semSignalB(s)` | 1 | **–1** | 0 |

```
/* program producerconsumer */
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();
          n--;
          m = n;
          semSignalB(s);
          consume();
          if (m==0) semWaitB(delay);
     }
}
void main()
{
     n = 0;
     parbegin (producer, consumer);
}
```

**Figure 5.10 A Correct Solution to the Infinite-Buffer Producer/Consumer Problem Using Binary Semaphores**

```
/* program  producerconsumer */
semaphore n = 0;
semaphore s = 1;
void producer()
{
     while (true)
     {
          produce();
          semWait(s);
          append();
          semSignal(s);
          semSignal(n);
     }
}
void consumer()
{
     while (true)
     {
          semWait(n);
          semWait(s);
          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 n= 0;
semaphore e= sizeofbuffer;
void producer()
{
     while (true)
     {
          produce();
          semWait(e);
          semWait(s);
          append();
          semSignal(s);
          semSignal(n)
     }
}
void consumer()
{
     while (true)
     {
          semWait(n);
          semWait(s);
          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**