Bounded Buffer Problem without Synchronization

This handout demonstrated the problem with process interaction when no synchronization is performed. This problem is called the *Bounded Buffer Problem* or the *Producer/Consumer Problem*. The *producer* process writes items into a finite buffer, and the *consumer* process reads them. All the variables are shared.

First, the shared variables:

```
struct item buffer[n];
int in, out, counter;
```

buffer is the shared buffer. *counter* is the number of elements currently in the shared buffer. *in* is the index of the element into which the next item is to be placed, and *out* is the index of the element from which the next item is to be removed.

Second Proposed Solution

Here, *inCS*[0] is **true** when process 0 is in the critical section, and **false** otherwise. A similar statement holds for *inCS*[1].

Now, the producer process code; we only list the code that operates on the shared variables.

```
while(1) {
           p = generate_item();
           while (counter == n)
                      /* do nothing */ ;
           buffer[in] = p;
           in = (in + 1) % n;
           counter++;
}
```

The consumer process code is similar.

```
while(1) {
           while (counter == 0)
                      /* do nothing */ ;
           p = buffer[out];
           out = (out + 1) % n;
           counter--;
}
```

If each loop is executed separately, these processes work as expected. But if they are intermingled, the result may be incorrect.

As an example, suppose both processes try to alter *count* at the same time. Let's say the compiler compiled the statements into the following:

```
P1 r1 = counter;            C1 r2 = counter;
P2 r1 = r1 + 1;             C2 r2 = r1 - 1;
P3 counter = r1;            C3 counter = r2;
```

Let *counter* be 3 when these are executed. Depending on how these statements are interleaved, the result of *counter* may be 2, 3, or 4:

C1 C2 P1 P2 C3 P3 results in *counter* being 4.

C1 C2 P1 P2 P3 C3 results in *counter* being 2.

C1 C2 C3 P1 P2 P3 results in *counter* being 3.

The problem is that two processes manipulated the variable *counter* simultaneously. Clearly, we need to ensure just one process does.