Producer-Consumer Problem

Note Title 19/10/2005

Producer Process

```
item nextProduced;

while (1) {
    while (((in+1) % Buffer_size) == out)
        ; // Do nothing
    buffer[in] = nextProduced;
```

```
    in = (in+1) % Buffer_size;
}
```

Start with in = 0 and out = 0

in = position produce an item in to

out = position to consume an item out of

Let's examine only the execution of the Producer first

in = 0 , out = 0 at the start

Buffer_size = 10

while((in+1) % Buffer_size == out) // While statement

(in+1) % Buffer_size = (0+1) % 10 = 1 % 10 = 1

out = 0 .

Clearly (in+1) % Buffer_size ≠ out

Since 1 ≠ 0 !!!

=> Skip NULL "Do Nothing" Statement

Execute next statement

buffer[0] = nextProduced;

↳ insert/produce ten into position 0 since in = 0.

Next statement

in = (in+1) % Buffer_size

=> in = (0+1) % 10 = 1.

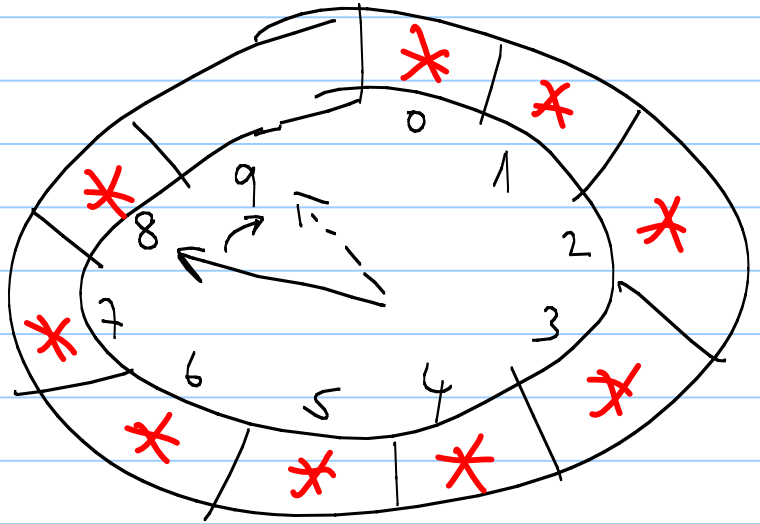
Note on modulo (%) :

$x \bmod y = x \,\%\, y$ = Value of Remainder after dividing x by y

Eg (i) $2 \,\%\, 10 \Rightarrow \frac{2}{10} = 0$ $R = 2$

(ii) $13 \,\%\, 10 \Rightarrow \frac{13}{10} = 1$ $R = 3$

Let's assume Producer produces items, and fills the buffer up to and including position number 8;



Once the Producer has finished producing item in position number 8, what are the values of in and out:

in = 9 (in = (in+1) % Buffer_size = (8+1) % 10 = 9)

out = 0 (ZERO)

Let's examine the "while" statement for another Run of Producer

while ((in+1) % Buffer_size == out)

; // Do nothing

(in+1) % Buffer_size = (9+1) % 10 = 10 / 10 = 0

But, out = 0

=> (in+1) % Buffer_size == out is TRUE

=> "Execute" the null statement ";"

and repeat the check of

(in+1) % Buffer_size == out

(= 0)

So, the Producer continually executes the "while + null statement", and does not produce an item in position 9, until ...

Busy waiting

... the consumer consumes item at position 0 (out → 1).

Let's examine the Consumer Process code now:

```
item nextConsumed;

while (1)
{
    while (in == out)
        ;   // Do nothing (Busy Wait)

    nextConsumed = buffer[out];

    out = (out + 1) % Buffer_size;
}
```

Reminder:

Values of our variables Before Consumer starts:

(after the Producer has run as described above)

in = 9

out = 0

Let's examine Consumer's while statement

```
while (in == out)
    ; // Do Nothing
```

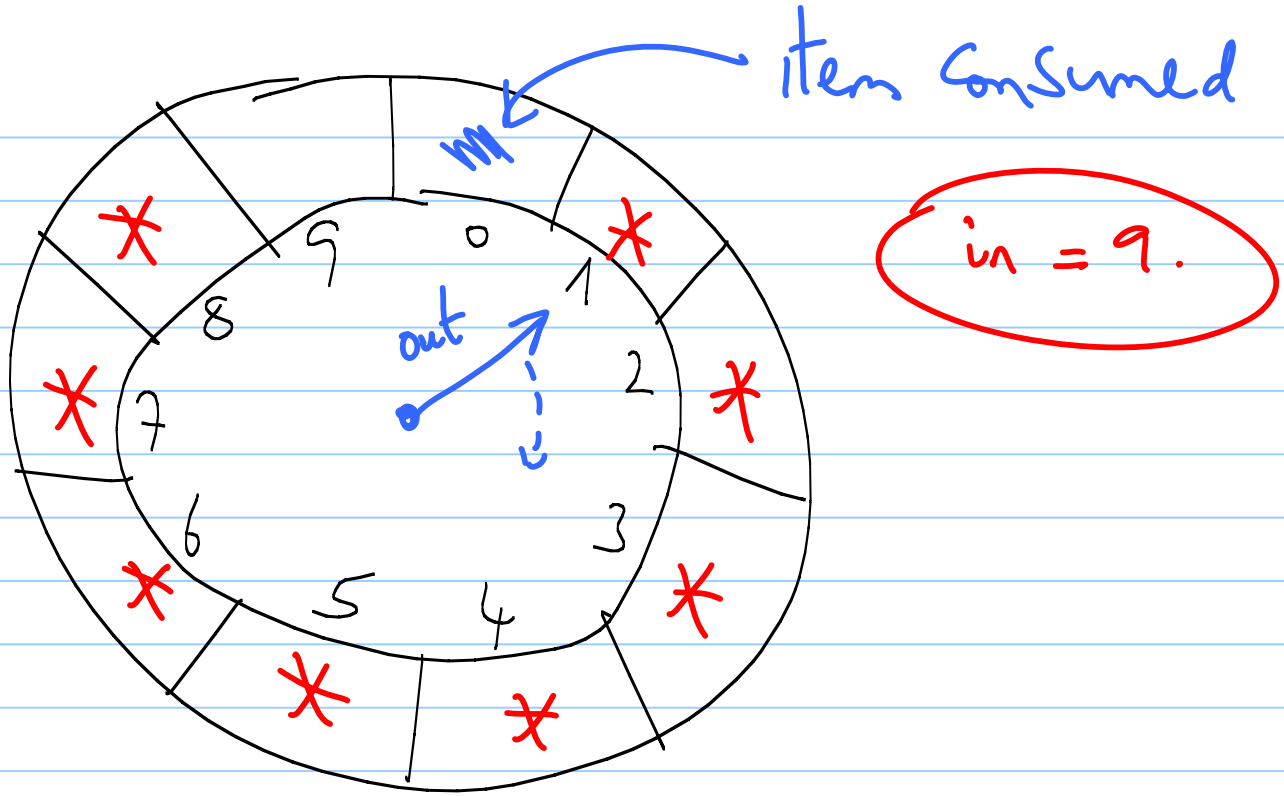
Since in = 9 and out = 0

clearly in ≠ out => skip NULL statement

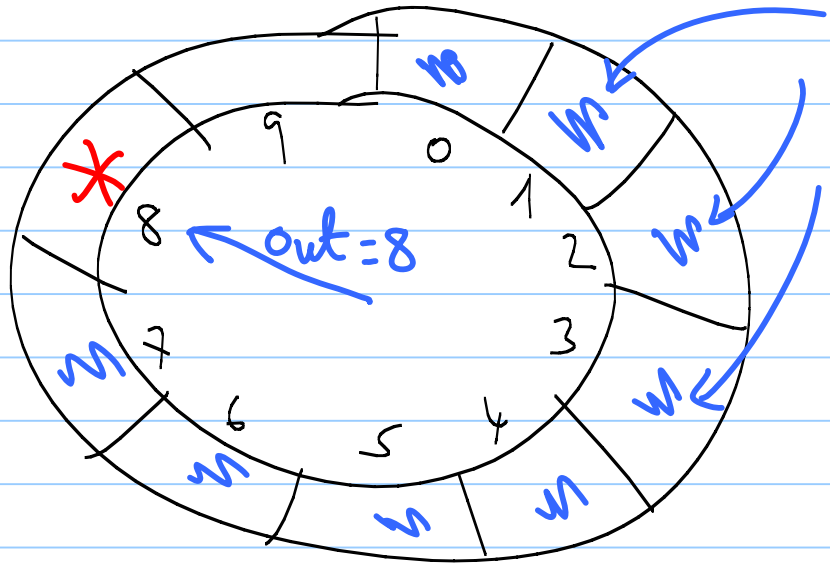
=> go to next statements:

```
itemConsumed = buffer[out]; // Consume item at 0
out = (out + 1) % Buffer_size
```

Picture now is as follows:


item consumed
in = 9.
out
0
1
2
3
4
5
6
7
8
9

So, let's consider the case where the Consumer consumes items, 0, 1, 2, 3, ..., 7, and is about to examine item at position 8



At this point our in and out variables are:

in = 9

out = 8

Examine Consumer's while statement:

```
while ( in == out )
        ; // Do nothing (Busy Wait)
```

Since in = 9 and out = 8

clearly in ≠ out => skipped NULL statement

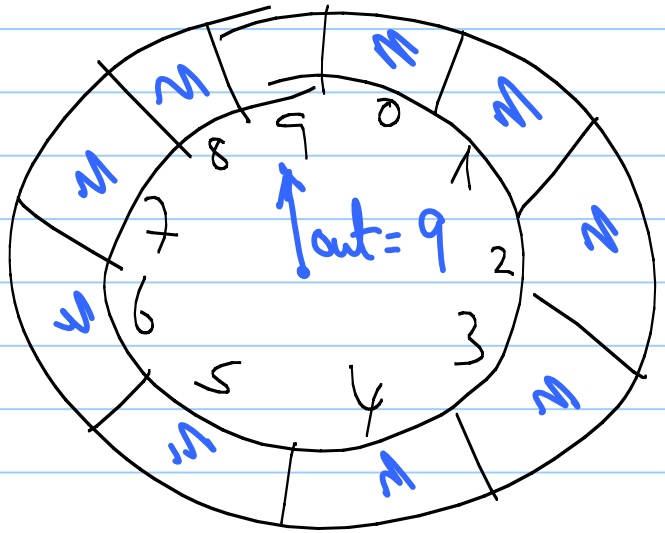
⇒ go on to next statements:

itemConsumed = buffer[8];

out = (out + 1) % BufferSize = (8+1) % 10

⇒ out = 9

Picture:



Buffer is empty (no items available for consuming).

Consumer executes again ...

```
while (in == out)
        ; // Do nothing
```

in = 9
out = 9

} ⇒ in does indeed = out

⇒ Do nothing! ";" statement (Busy Wait).

WHAT WE HAVE SHOWN:

1. The code for Producer & Consumer Processes allows us to fill only (Buffer_size − 1) spaces ... (uneconomical / inefficient)

2. Code prevents Producer and Consumer from accessing the same buffer space at the same time.

(3) Producer busy-waits when (Buffer_size − 1) spaces have been filled

(4) Consumer busy-waits when the buffer is empty

We say that the Solution (code) is correct because the code prevents data inconsistencies.