Producer-Consumer Problem

Consumer

```
void consumer(void)
{
 int item c;
  while (true){
 while (count==0);
 item c = buffer(out);
 out = (out+1) mod n;
 count = = count -1
 Process-item(item c)
```

1. load Rc, m[count]
2. DECR, Rc
3. Store m[count], Rc

gets from buffer

Producer

```
        int count = 0
void producer (void) {
   int item p;
   while (true) {
      produce item (item p);
      while (count==n);
      Buffer[in] = item p
      in = (in+1) mod n
      count = count+1;
   }
}
```

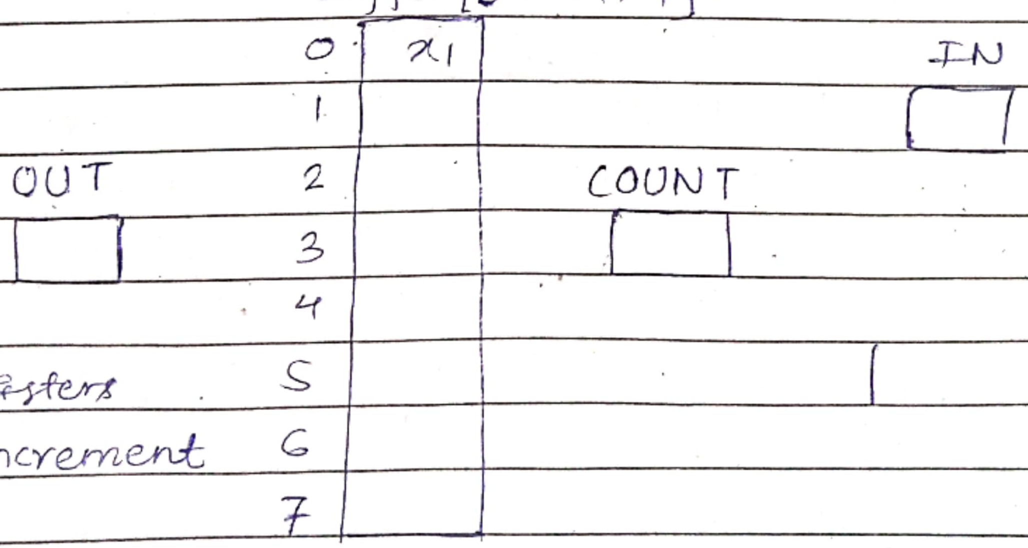
~~1. load R~~

gives to buffer

1. load Rp, m[count]
2. INCR Rp.
3. Store m[count], Rp

n = 8 ✓.

Buffer[0 . . . n-1]



Rp → Registers

INCR → increment

n = size of buffer

Case I: x_1 (producer is processing x_1)

Out	$(0+1)$ mod 8	in	count
$\cancel{0}$ 1	1 mod 8 = 1	$\cancel{0}$ 1	$\cancel{0}$ $\cancel{1}$ 0

$(0+1)$ mod $8 = 1$
count $= 1-1 = 0$

It forms a circular queue so we used ~~a~~ mod n.

- → ~~I.~~ Firstly the producer produces a item
- → Starting position of in is 0. in tells the address of the next empty slot:
- → So we fill buffer[in] i.e. buffer[0] with value x_1 & in is incremented to 1
- → Initially the count is zero & we change it (i.e. we increment it) to 1.

- → Out also starts from zero & points to the next item to be consumed
- → Now x_1 comes in item C
- → Out is incremented to 1 & count is decremented to 0.

Its the best case in ~~while~~ which producer comes before consumer.

Case 2 : (Producer has produced 3 items)

x_1, ~~&~~ x_2, x_3

So now,

Out	In	Count
~~0~~ 1	~~3~~ 4	~~3~~ ~~4~~ 2

→ Now first the producer produces an item x_4

→ As In is ~~$~~3, it's stored at pos. 3

→ In incremented to 4 & ~~count to 4~~

0	x_1
1	x_2
2	x_3
3	x_4
4	
5	
6	
7	

→ Now while incrementing count, consider the code given behind,

→ First R_p holds the value 3, ~~&~~

→ R_p gets incremented 4

→ But consider that producer gets pre-empted now, due to any reason

→ Now we check whether there's any other process in ready queue, so there's consumer

→ So ~~we~~ we go to the consumer, as the count != 0

→ So we store buffer(0) in count & increment out

→ Now while decrementing the count, we store count in R_c

→ Decrement it

→ So count value is 3, which is stored in Rc & got decremented to 2. ($\therefore R_c = 2$).

→ Now consumer get pre-empted & Producer comes again

→ So ~~we as~~ we know producer has been executed till I_2 which is stored in PCB, so we'll resume it.

→ So, the value of Rp which is 4 gets loaded into count & producer code gets terminated & control goes to consumer

→ Consumer now performs the ~~fourth~~ third remaining instruction & loads value 2 in count (which means that there are 2 items in buffer) but actually 3 are there; so count is wrong.

~~Prot~~ Case:

(Producer) I_1 I_2 (Consumer) I_1 I_2 (Prod.) I_3 (Con.) I_3

So race condition has occurred

Solution to Producer Consumers problem using Binary Semaphores

Counting Semaphores → full = 0 = No. of filled slots

→ Empty = N = No. of empty slots

```
Producer item(itemp)                 Consumer
1) down(empty);                      1) down(full);
2) down(S);                          2) down(S);
3) Buffer[IN]=itemp                  3) itemc=Buffer[out]
4) in=(in+1) mod n                   4) out=(out+1) mod n
5) Up(S)                             5) Up(S);
6) Up(full)                          6) Up(Empty)
```

Case 1: Consider the foll. scenario (Without Context switching)

N = 8

in [~~3~~] 4

Out [0]

0	~~a~~
1	b
2	c
3	d
4	
5	
6	
7	

Empty = ~~8~~ ~~4~~ 5   S = ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1

full = ~~3~~ ~~4~~ 3

→ Let's say Producer comes first & does down of empty to 4 & S (1→0)

→ Buff [3] = d   In = ~~4~~3 → 4

full = (3→4)

→ S again goes to 1 after producer exits

→ Now consumer enters

full (4 → 3)   ~~S (1~~ = S(1→0)

itemc = a   out (0→1)

S(0→1)   Empty (4→5)

No problem here.

Case 2 :

N = 8

0	~~a~~
1	b
2	c
3	d
4	
5	
6	
7	

In ~~3~~ 4

Out ~~0~~ 1

S = ~~1~~ ~~0~~ ~~1~~ ~~0~~ 1

Empty = ~~5~~ ~~4~~ 5

full = ~~3~~ ~~2~~ 3

→ Producer comes first

Empty (5→4)

→ Before down of S producer is pre-empted

→ Consumer comes

→ full (3 → 2)   S (1 → 0)

→ out (0 → 1)

→ S (0 → 1)   → Empty (4 → 5)

→ Producer comes

S (1 → 0)

Buff [3] = d

Once either producer or cons. has entered CS. the other one can't enter CS as S value would be zero

In (3 → 4)   full (2 → 3)

S (0 → 1)

∴ We can see no problem is encountered