Producer

Count (Shared Variable) = 0

Consumer

- Count != Buffersize
- Buffer = Product
- Count = count+1

BUFFER MEMORY

Shared memory

Producer - Consumer Problem

Count → no. of products in Buffer

Condition at Producer Side (Count will be incremented)

* Before placing product into Buffer Producer have to check Buffer is not FULL

Condition at Consumer Side (count will be decremented)

* Before consuming product from Buffer Consumer have to check Buffer is not EMPTY

Producer

Count (Shared Variable) = 0

- Count != Buffersize
- Buffer = Product
- Count = count+1

BUFFER MEMORY

Shared memory

Consumer

- Count != 0
- Consumed = product
- Count = Count - 1

Producer	Consumer
I_1 : R_1 = count	I_4 : R_2 = Count
I_2 : $R_1 = R_1 + 1$	I_5 : $R_2 = R_2 - 1$
I_3 : Count = R_1	I_6 : Count = R_2

Instruction Execution

let us assume count = 4

I_1 : $R_1 = count \Rightarrow R_1 = 4$

I_2 : $R_1 = R_1 + 1 \Rightarrow R_1 = 5$

I_4 : $R_2 = count \Rightarrow R_2 = 4$

I_5 : $R_2 = R_2 - 1 \Rightarrow R_2 = 3$

I_6 : $count = R_2 \Rightarrow count = 3$

I_3 : $count = R_1 \Rightarrow count = 5$

Actual

count = 4

producer → product

count = 5

Consumer consumes

count = 4

1. m (mutex), a binary semaphore which is used to acquire and release the lock.
2. empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3. full, a counting semaphore whose initial value is 0.

Producer

```
do {
  wait (empty); // wait until empty>0
                and then decrement 'empty'
  wait (mutex); // acquire lock
  /* add data to buffer */
  signal (mutex); // release lock
  signal (full);  // increment 'full'
} while(TRUE)
```

1. m (mutex), a binary semaphore which is used to acquire and release the lock.
2. empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3. full, a counting semaphore whose initial value is 0.

Producer

```
do {
  wait (empty); // wait until empty>0
                   and then decrement 'empty'
  wait (mutex); // acquire lock
  /* add data to buffer */
  signal (mutex); // release lock
  signal (full);  // increment 'full'
} while(TRUE)
```

Consumer

```
do {
  wait (full); // wait until full>0 and
                  then decrement 'full'
  wait (mutex); // acquire lock
  /* remove data from buffer */
  signal (mutex); // release lock
  signal (empty);  // increment 'empty'
} while(TRUE)
```