is of finite size

* Bounded Buffer Producer consumer Problem

1.) Producer Process ⇒ To Produce an item and put it into a buffer of finite size.

2.) consumer Process.
⇓
To consume the item already placed in a buffer.

→ Producer cannot keep the item which is already full.

→ Consumer can't consume an item from an empty buffer.

* Buffer needs to be checked by both the producer and consumer.

→ Shared item b/w the producer and consumer.

→ Variable related to the buffer is shared among producer and consumer.

∴ Buffer is implemented as circular queue

- Var 1
- Var 2

↑ Modulus operator to check the status of the buffer.

∴ Producer

↳ Possible check for the space.

↳ counter = counter+1 ;

↳ Put the element in the buffer

∴ consumer

↳ Possible checking for empty

↳ consume item ~~[illegible]~~

↳ counter = counter-1;

→ counter is a shared variable.

q →

```
r1 := counter            CPU → r2 := counter
r1 := r1 +1 ;                  r2 := r2 - 1 ;
counter := r1 ;                counter := r2 ;
```

⇓

Critical section.

3

Process Synchronization

When two or more process execute concurrently then suitable means to be provided for execution of code involving a shared resource.

Producer	consumer
counter = counter +1	Counter = counter -1
r_1 = counter;	r_2 = counter ;
$r_1 = r_1 + 1$;	$r_2 = r_2 - 1$;
counter = r_1	counter = r_2

↳ critical section
↳ critical section Problem.
↳ Protocols for the sol^n of CSP
↳ $cond^n$ for the sol^n of CSP.
↳ Two process S/w. Sol^n (Peterson sol^n)
↳ Drawback of

$P_1, P_2, \text{---} P_n \rightarrow n$ number of Processes. executes concurrently.

R → shared Resource.

Section of code in each of the processes involving the shared ResourceR known as critical section.

The specialization problem associated to the processes executing concurrently involving a shared resource R, known as critical section Problem.

↳ Protocol must be there for the solution of <u>critical section Problem.</u>

↳ It has to execute a section of code before entering into the critical section → entry section

↳ upon completion of critical section it executes a section of code allowing the other to enter their critical section } exit section.

↳ Exit section may be followed by some code known as remainder section.

⁂ Conditions for the Solution of the critical Section Problem.

(1) Mutual exclusion :→

⇓

Processes are allowed to execute their critical section in a Mutually exclusive manner.

(2) Progress :→ If the critical section is ~~the~~ a free then it must be granted to a requesting process in finite time.

(3) Bounded waiting :→ Max. no. of time a process is allowed to enter its critical section.

Two Process Software solution.

P_0 , P_1

R

⁂ To develop Suitable code for entry section and exit section

```
i, j=i-1
        Pi
          repeat
                entry section
              critical section
                exit section
            remainder section
          until false.
```

Peterson Solⁿ (Two process s/w solⁿ)

```
Pi:    repeat
             flag[i] := true;
               turn := j
         while( flag[i] and turn=j) do skip;
       critical section
             flag[i] = false

             remainder section.
       until false.
```

1.) Pj is already executing

2.) Pj has expressed its interest to execute.