```
while (true)
{
// item not produced
while ((in+1) % BS = = out)
; // do nothing as the Buffer is Full
buffer [in] = next-produced-item;
in  = ( in + 1 ) % BS ;
}
```

```
while (true)
{
// item not consumed
while (in = = out)
; //do nothing as Buffer is Empty
next-consumed-item = buffer[out];
out  =  (out + 1 ) % BS ;
}
```

BS- Buffer Size

buffer - circular array with two pointers / access indices

in - next free position in the buffer array ;

out - first full position in the buffer array ;

in == out implies EMPTY; (in+1) % BS = = out implies FULL buffer

- Message passing – processes communicate without shared variables
- IPC facility provides two operations: - send(message) – message size fixed or variable and receive(message)
- Steps for IPC - Establish a communication link between processes and - Exchange messages via send/receive

Communication link types –

- **Direct or indirect ; Synchronous or asynchronous - Automatic or explicit buffering**

DIRECT

- Processes must name each other explicitly:
- send(P , message) – send a message to process P - receive(Q , message) – receive a message from process Q
  - Links are established automatically between the two processes
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

INDIRECT

- Messages sent / received from mailboxes (aka ports)
- Every mailbox has a id- Processes can communicate only if they share a mailbox
- Link established only if processes share a common mailbox
- A link may be associated with more than two processes
- Each pair of processes may share several communication links –
- Link may be unidirectional or bi-directional

Operations - Create a new mailbox - Send and receive messages through mailbox - Destroy a mailbox

- ✓ **send(A , message) –** send a message to mailbox A
- ✓ **Receive(A , message) –** receive a message from mailbox A

MAILBOX SHARING

Assume - P1, P2, and P3 share mailbox A - P1, sends; P2 and P3 receive ; Issue to resolve who gets it ?

- link to be associated with at most two processes
- only one process at a time to execute a receive operation
- system arbitrarily selects receiver and sender is informed of the same. Sender is

SYNCHRONIZATION

- ✓ Message passing 2 types - blocking ; non-blocking
- ✓ Blocking =synchronous and Non blocking = asynchronous
- Blocking send - sender blocked until the message is received
- Blocking receive -receiver blocked until a message is available
- Non-blocking send -sender sends message & continues
- Non-blocking receive -receiver receives a valid message or null

Messages are always queued (Buffer)

3 types of Buffers

(1) Zero capacity – queue has maximum length of 0

- Sender must wait (or block) until the receiver gets the message

(2) Bounded capacity – queue has finite length of n messages
Sender must wait if link full

(3) Unbounded capacity – queue has 'infinite' length ; Sender never waits

- ✓ POSIX library has support for various interfaces to perform shared memory such as shmat, shmdt,shmctl, etc. to support Inter Process Communication.
- ✓ We will visit SHM mode of IPC post PIPES to be explored first for its convenience ; code simplicty ; synchronous nature
- ✓ SHM is best for async and multiple process setup

PIPES for IPC

- Act as conduit to allow process to communicate
- Simpler mechanism for IPC
- Issues to be addressed : Bi or Uni directional communication
- Simplex – data can travel only in one direction (source and destination ends are fixed)
- Half Duplex – data can travel only in 1 direction at a time
- Full Duplex – both directions at a time possible
- What sort of relationship (P-C) between communicating processes
- Can pipes communicate over a network
- Ordinary Pipes – 2 Processes communiate in standard Producer Consumer Fashion

PIPES for IPC

- Ordinary Pipes – 2 Processes communicate in standard Producer Consumer Fashion
- Producer writes at one end of the pipe (write end)
- Consumer Reads from other end of the pipe (read end)
- Ordinary pipes are unidirectional – One way communication
- Simulate Bi directional communication with two pipes – with each sending data in different direction
- Process writes message onto pipe which is read by other processes.
- System call – pipe (int fd[2]); // function declaration
- fd[0] – read end of the pipe ; fd[1] – write end of the pipe
- fd - file descriptor

PIPES for IPC

- Simplex – one direction transfer only at a time ;
- Half Duplex – Both directions data exchange possible but not both at the same time
- Full Duplex – Both directions data exchange at the same time possible
- Ordinarily Pipes cannot be accessed from outside the process that created it
- Ideal setup – Parent creates pipe and used to communicate with the child process
- Child process inherits pipes from parent process