* Inter process communication enables and facilitates communication between processes

* Xinu uses message-passing (Pipes, Socket etc)

* Different processes / Different address spaces.

* Message passing design decisions

1. Size of message
2. Fixed or variable size
3. Maximum size of message
4. Synchronous or Asynchronous
5. Buffer size: how many outstanding messages can be.
6. Where messages are stored?
7. How is recipient specified? (Receiver knows who is sending it the message)
8. How does receiver know the sender's identity?
9. Replies supported?

Q) Why message passing mechanism is difficult to design?

1. Processes need to coordinate
2. Memory management -> allocation of memory for the kernel.

③ Affects several subsystem

- * Direct process-to-process communication Not one-to-many
- * Only supports one-word messages (32 bits)
- * Messages stored with receiver.

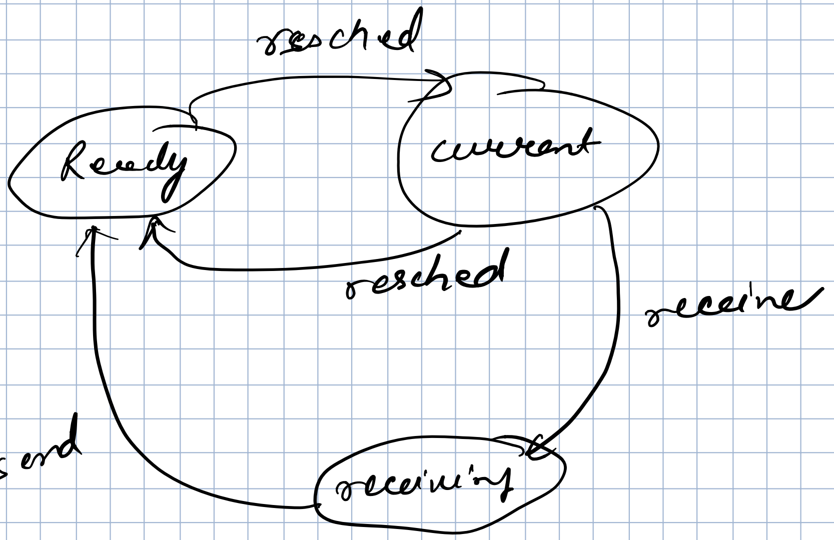
  Kernel side. Buffer is present at the receiver side

- * One-message buffer.
- * Synchronous
- * Three system call in Xinu :- (non-blocking) \xrightarrow process id of receiver
  1. send (pid, msg)
  2. msg = receive()
  3. msg = recvclr() → returns message and clear buffer. (non-blocking)
- * Pipes synchronous in both ends.
- * Messages stored in receiver's process table entry.
- * Send transmits message to specified process.
- * Receiver blocks until message arrives.
- * Recvclr removes existing message in buffer.
- * First message semantics

One message at a time - First message

one message at a time. First message sent is stored until it has received. Subsequent attempts to send fail.

* what should we do inside the kernel when we call-wait in terms of process state?



* The receiver unblocks itself only when receiver receives a message.

Send

* prhasmsg (field in process table)

prptr = &proctab[pid]
↳ receiver's id

bool 8 (Non-zero if msg is valid)

* prmsg (buffer for message)

* If there already a message in the

buffer, we return an error