Message Passing

DESIGN ISSUES IN IPC BY MESSAGE PASSING

- SYNCHRONIZATION
  - BLOCKING PRIMITIVE (SYNCHRONOUS) : If its invocation blocks execution of its invoker
  - NON-BLOCKING (ASYNCHRONOUS) : Does not block execution
- 2 cases
- Ist case
  - Blocking send
  - Blocking Rec
- 2nd case
  - Non-Blocking Send
  - Non-Blocking Rec

ISSUES IN BLOCKING SEND

- Sending process could get blocked forever if receiving process crashes or message lost on the network due to communication failure
- Hence it should use timeout values
- It could be a parameter of “send” primitive

ISSUES IN BLOCKING RECEIVE

- Receiver process could get blocked forever if sending process crashes or message lost on the network due to communication failure

- Hence it should use timeout values

ISSUES IN NON-BLOCKING RECEIVE

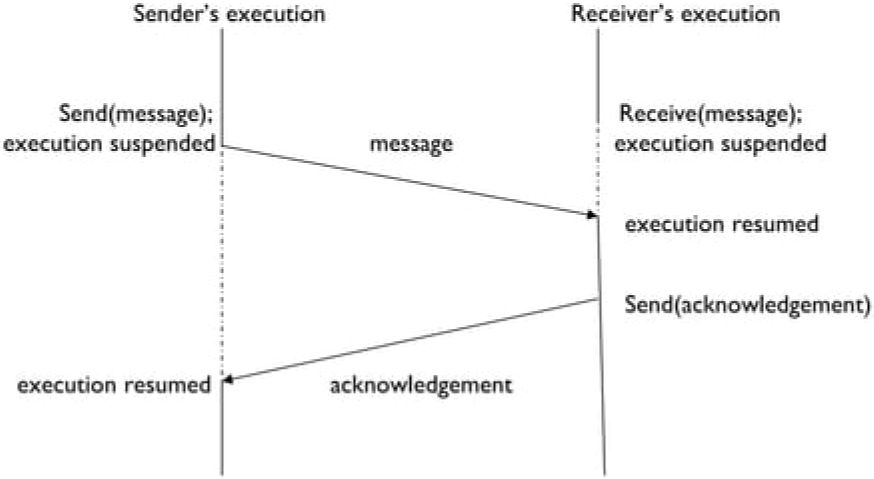
- How the receiving process knows that the message has arrived in the message buffer?

- Done by Polling or Interrupt

    - POLLING: Receiver uses “test” primitive to allow it to check buffer status

    - INTERRUPT : When message has arrived in the buffer, a s/w interrupt is used to notify the receiving process

SYNCHRONOUS COMMUNICATION



Synchronous mode of communication with both send and receive primitives having blocking-type semantics

IMPLEMENTATION EASE

- Synchronous communication is easy to implement

- If message gets lost or is undelivered, no backward error recovery is required

- Synchronous communication limits concurrency

- Subject to communication deadlock

BUFFERING

- Messages are transmitted from one process to another by copying the body of the message from the address space of sending process to address space of receiver process (possibly via address space of kernels of sending and receiving computers)

- In some cases, the receiver process may not be ready to receive a message but may want O.S. to save messages for later reception

- Message buffering strategy related to synchronization strategy
    - Synchronous Mode: Null /No Buffer
    - Asynchronous Mode: Buffer with unbounded capacity

Null Buffering

- It has two strategies

1st strategy

- Message remains in SPAS (Sender Process Address Space) and the execution of send is delayed until the receiver executes the corresponding receive
- After send, when ACK is received, it executes “send” again

2nd strategy

- The message is simply discarded and the time out mechanism is used to resend message after a time out period
- After executing send, sender process wait for an ACK

Single-Message Buffer

- If the receiver is not ready, a message has to be transferred two or more times hence null buffer strategy is generally not suitable for synchronous communication

- Therefore, synchronous communication use a single-message buffer strategy

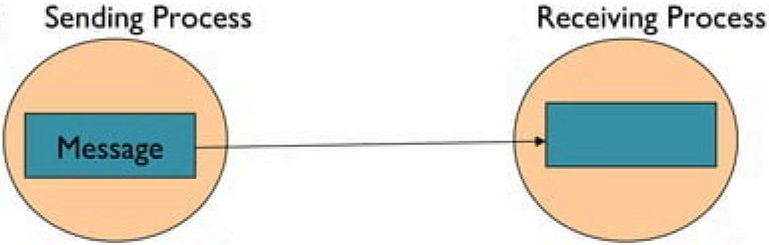
- A buffer having a capacity to store a single message is used on the receiver's node

FINITE BOUND BUFFER

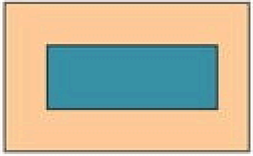
- A create-buffer system call is provided to the user.

- This system call when executed by a receiver process creates a buffer of a size specified by receiver either in kernel AS. or receiver process AS.

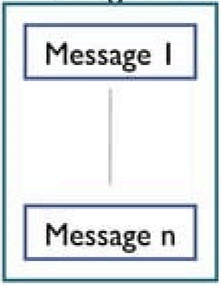
i. Null buffer:



ii. Single Message Buffer:



iii. Multiple Message Buffer:



ENCODING AND DECODING OF MESSAGE DATA

Structure of program/object should be preserved while they are being transmitted from address space of sending process to RPAS.

Difficult to achieve because

- Absolute ptr losses its meaning when transferred from one process AS to another
  - Requires flattening and shaping of objects
- Receiver process should have a-priori knowledge of varying memory occupied by various data items

Due to above problems, program object not transferred in their original form. First converted to stream form by encoding.

Two representation may be used for encoding and Decoding

1. Tagged Representation: Data object along with its type is encoded
2. Untagged Representation: No information regarding data type.

- Untagged: SUN XDR (Extended Data Representation)
- Tagged: A.S.N (CCITT 1985)

PROCESS ADDRESSING

- Process addressing is naming of parties involved in interaction
- Two types of process Addressing:

1. Explicit Addressing: The process with which communication is desired is explicitly named as a parameter
   - Send (process-id, message)
   - Receive (process-id, message)

Implicit Addressing

Does not explicitly name a process for communication

- Send_any (service_id, message)
- Send a message to any process that provides the service of type "service_id"

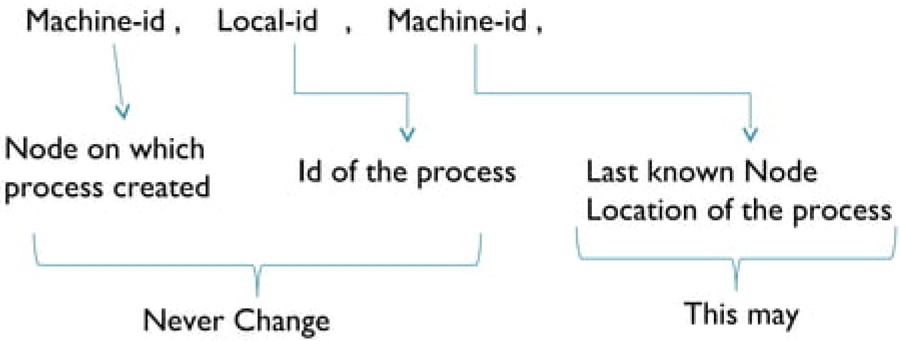
- Receive_any (process_id, message)
- Receive a message from any process and returns the "process_id" of the process from which the message was received.

PROCESS ADDRESSING (contd..)

- Simple method to identify a process is by a combination of machine-id and local-id such as machine-id@local-id.
- Local-id can be process- id or port-id, that uniquely identifies a process on a machine.
- Machine-id is used by sending machine kernel to send the message to the receiver process machine.
- Eg: Berkely UNIX -32 bit Internet address for machine-id
  -16 bit for local-id

- **Drawback:** It does not allow a process to migrate from one machine to another in the case of heavy load.

I. To overcome this limitation



Machine-id , Local-id , Machine-id ,

Node on which process created

Id of the process

Last known Node Location of the process

Never Change

This may

This type of adding is known as link based addressing.

During Migration, a link Information (p-id + m/c id of new node) of the new node is left on previous node.

Drawbacks: i. Overload of locating a process if process has migrated several times.

ii. It may not be possible to locate a process if an intermediate node on which the process once residing during its lifetime is down.

TWO LEVEL NAMING SCHEME FOR PROCESSES

- Each process has two id:
  - A high level name that is m/c independent (ASCII string)
  - A low level name that is m/c dependent (m/c id@local id)
- A name server is used to maintain a mapping table.
- Now when a process wants to send a message to another process, it specifies high level name of the receiver process.

TWO LEVEL NAMING SCHEME FOR PROCESSES

- The kernel of the m/c first contacts the name server to get low level name
- Can also be used for FUNCTIONAL ADDRESSING

  High level name identifies a service instead of a process
- Precaution of Replicating Name Server.

FAILURE HANDLING

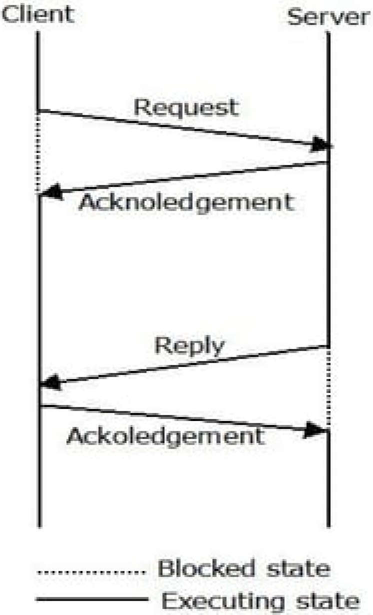
- Loss of Request message
- Loss of Response Message
- Unsuccessful execution of request

To overcome these problems

- A reliable IPC protocol of a message-passing system is designed.
- It is based on the ideas of internal retransmission of messages after time out and
- Return of an ACK to sending m/c kernel by receiver m/c kernel.

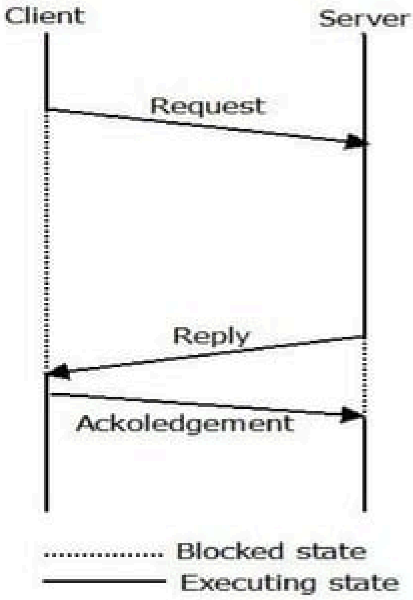
- The time for which the sender waits is slightly more than the approximate round trip time + the average time required for executing the request

1. FOUR MESSAGE RELIABLE IPC Protocol



The four-message reliable IPC protocol for client-server communication between two processes.

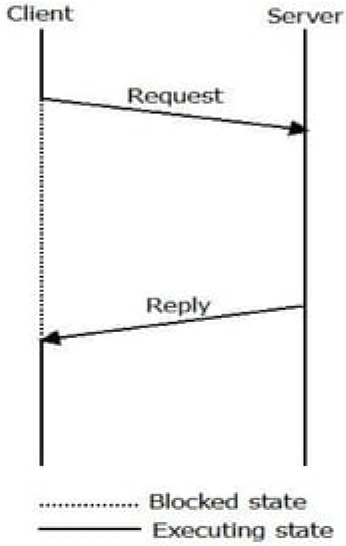
2. THREE MESSAGE RELIABLE IPC



The three-message reliable IPC protocol for client-server communication between two processes.

- What happens when request processing takes a long time?
- **Answer:** Server sends a separate ACK to acknowledge request message

3. TWO MESSAGE RELIABLE IPC



The two-message IPC protocol used in many systems for client-server communication between two processes.

- When request received at servers machine, it starts a timer
- If server finishes processing the req. before time expires, reply acts as ACK
- Else a separate ACK is sent by kernel

Idempotency

- Means repeatibility
- An Idempotent operation produces the same result without any side effects, no matter how many times it is performed with the same argument.
- ISSUE : Duplicate Requests

Handling of Duplicate Requests

- If client makes a request
- Server processes the request
- Client doesn't receive the response
- After time out, again issues REQ

- What Happens?

Handling of Duplicate Requests

- Use unique id for every REQ
- Kernel on the server maintains a reply cache

Keeping Track of Lost and Out-of-Sequence Packets in Multidatagram Messages

- Stop n Wait Protocol
  - ACK for each packet
- Blast Protocol
  - Single ACK for all packets of multidatagram message
  - Two fields in each packet – total no. of packets and seq no. of packet
  - After timeout – Selective Repeat

Group Communication

- A group is a set of parties that, presumably, want to exchange information in a reliable, consistent manner.
- Group communication is a paradigm for multi-party communication that is based on the notion of *groups* as a main abstraction.
- For example:
  - The participants of a message-based conferencing tool may constitute a *group*.
  - If one message is a response to another, the original message should be delivered before the response.

Group Communication

The set of replicas of a fault-tolerant database server may constitute a group.

- Consider update messages to the server. Since the contents of the database depend on the history of all update messages received, all updates must be delivered to all replicas. Furthermore, all updates must be delivered in the same order. Otherwise, inconsistencies may arise.

Group Communication

- Following three types of group communication are popular:
    - One to Many
    - Many to One
    - Many to Many

Message Delivery to Receiver Process

- User applications use high level group names in programs
- Centralized group server (GS) maintains a mapping of high-level group names to their low level names
- Group server also maintains a list of process identifier of all the processes for each group

Message Delivery to Receiver Process

- When a sender sends a message to a group specifying its high level name
- Kernel contacts the GS to obtain low level name & p_id of processes belonging to that group
- This list of p_id is inserted in the message

Buffered / Unbuffered Multicast

- Multicast is a Asynchronous operation?
- So which one to use
  - BUFFERED or UNBUFFERED?

Reliability in Multicasting

- Depends on degree of reliability required
- Sender of a multicast message can specify the number of receivers from which a response message is needed
- This is expressed in the following form:
  - 0-reliable
  - 1-reliable
  - ‘m’ out of ‘n’ reliable
  - all reliable

Atomic Multicast

- This has all or nothing property i.e. when a message is sent to a group, either all or none receive it.

- Only “all-reliable” kind of reliability needs this strict paradigm

Group Communication Primitives

Group communication is implemented using middleware that provides sets of primitives to the application.

- **Multicast primitive** (e.g., post): This primitive allows a sender to post a message to the entire group.

OR

send() / send_group(): for “1 – 1” and “1 - m” semantic

Name Server or Group Server?

Group Communication Primitives

- **Membership primitives**
- e.g., join, leave, query_membership;
- These primitives allow a process to join or leave a particular group, as well as to query the group for the list of all current participants.

Mnay to One Comm. Issues

- The single receiver may be selective or nonselective
    - Selective – Deterministic
    - Non-selective – Non-Deterministic

Many to Many Comm. Issues

- One to many and many to one implicit in this scheme
- Issue of ordered message delivery
- Semantics of ordered delivery are:
  - Absolute Ordering
  - Consistent Ordering
  - Causal Ordering

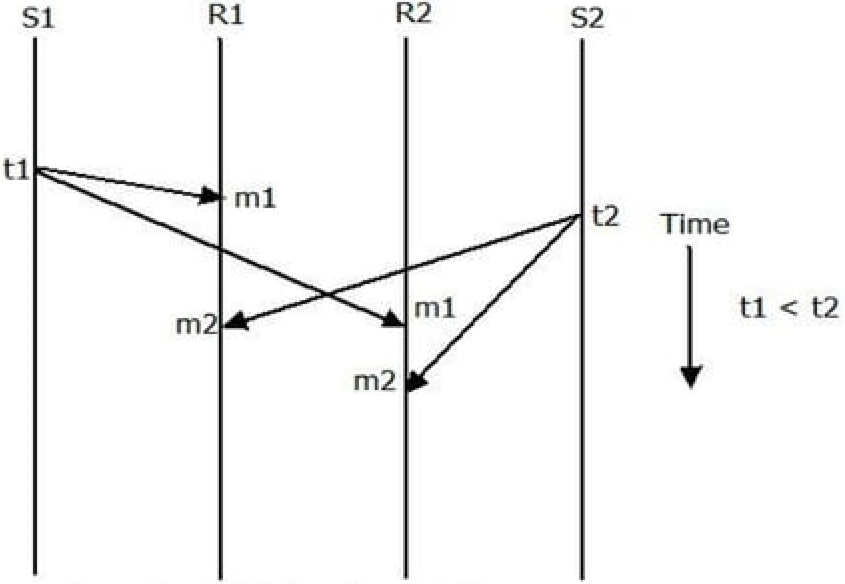
Absolute Ordering

- All messages are delivered to all receiver processes in the exact order in which they were sent

- System is assumed to have a clock at each machine and all clocks are synchronized with each other

- Here clock value is taken as the identifier of the message

Absolute Ordering

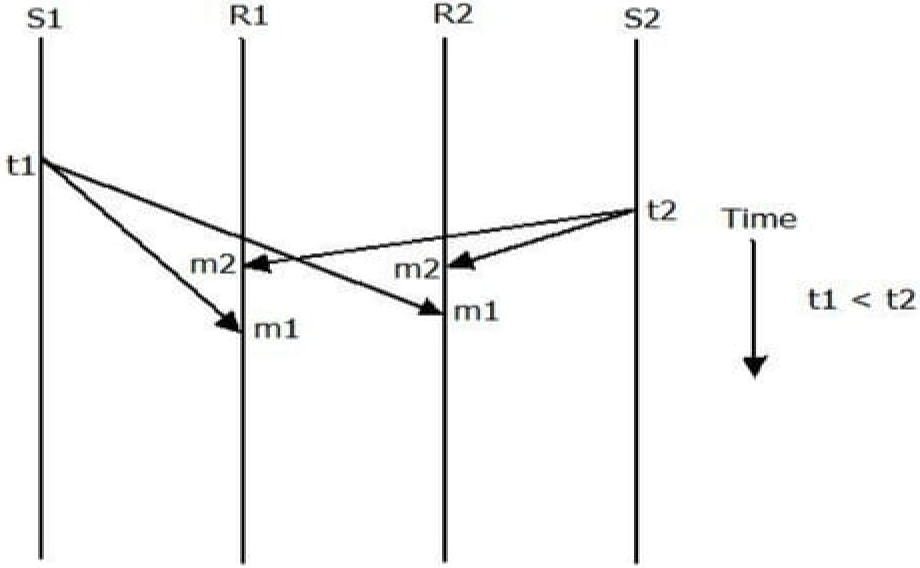


Absolute ordering of messages

Consistent Ordering

- Absolute ordering requires globally synchronized clocks, which are not easy to implement
- Consistent ordering ensures that all messages are delivered to all receiver processes in the same order
- Order may be different from the order in which messages were sent
- Sending machines send messages to a single receiver (*sequencer*) that assigns a sequence number to each message and then multicasts it
- Subject to single point of failure and hence has poor reliability

Consistent Ordering



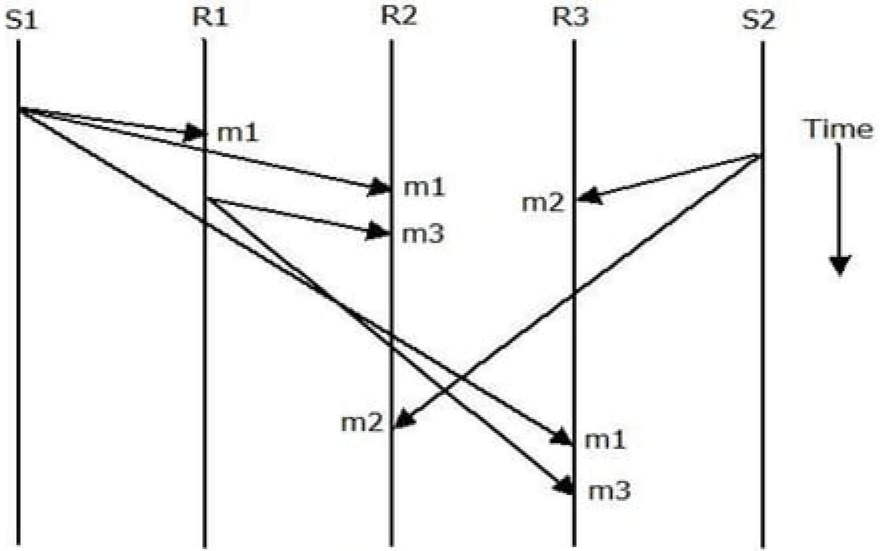
Consistent ordering of messages

Causal Ordering

- This ensures that if the event of sending one message is causally related to the event of sending another message, the two messages are delivered to all receivers in the correct order

- Two message sending events are said to be causally related if they are corelated by happened-before relation

Causal Ordering



Causal ordering of messages