GOVERNMENT COLLEGE OF ENGINEERING, JALGAON
(An Autonomous Institute of Government of Maharashtra)
Department of Computer Engineering

Name: Vaibhav Bharat Sontakke	PRN No: 1942207
Class: T. Y. B. Tech.	Batch: L4
Date of Performance: __________	Date of Completion: ____________
Subject: DSL	Sign. of Teacher with Date: __________

Experiment No.: 03

Aim: To Study Message Passing Interface (MPI)

Title: To Study Message Passing Interface (MPI) for Distributed Computing system.

Theory :

Message Passing Interface (MPI)

Objective:

Message Passing Interface (MPI) is a standardized and portable message passing system distributed and developed for parallel computing. MPI provides parallel hardware vendors with a clearly defined base set of routines that can be efficiently implemented.

The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.

In parallel computing, multiple computers -- or even multiple processor cores within the same computer -- are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks.

FEATURES:

1. General
   - Communicators combine context and group for message security-Thread safety

2. Point-to-point communication
   -Structured buffers and derived datatypes, heterogeneity

   - Modes: normal (blocking and non-blocking), synchronous, ready (to allowaccess to fast protocols), buffered

3. Collective
   - Both built-in and user-defined collective operations
   - Large number of data movement routines
   - Subgroups defined directly or by topology
4. Non-message-passing concepts not included:
   - process management
   - remote memory transfers
   - active messages
   - threads
   - virtual shared memory
5. MPI does not address these issues, but has tried to remain compatible with theseideas (e.g. thread safety as a goal, intercommunicators)

<u>**Working of MPI in Distributed Computing**</u>:

- **Distributed Memory**

  Every processor has its own local memory which can be accessed directly only by its own CPU. Transfer of data from one processor to another is performed over a network. Differs from shared memory systems which permit multiple processors to directly access the same memory resource via a memory bus.

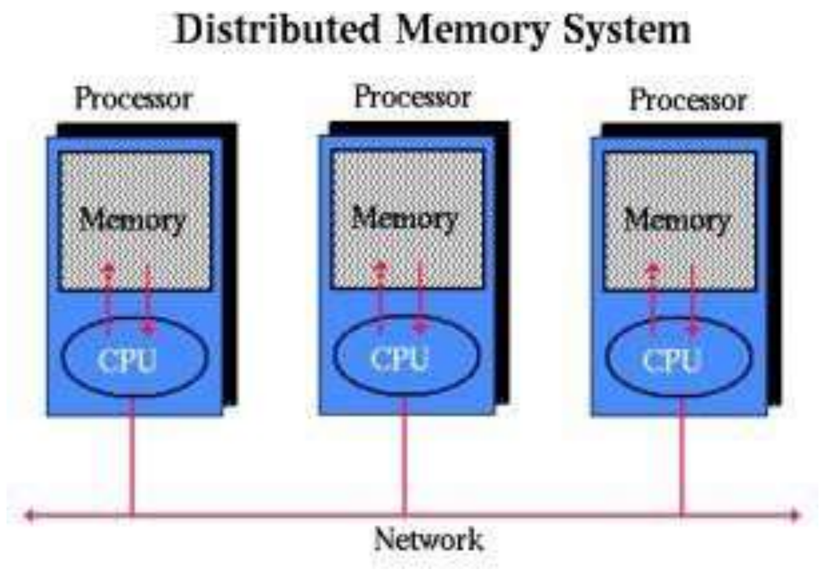


Figure: Message Passing Interface in DS

- **Message Passing**

  The method by which data from one processor's memory is copied to the memory of another processor. In distributed memory systems, data is generally sent as packets of information over a network from one processor to another. A message may consist of one or more packets, and usually includes routing and/or other control information.

- **Processor**
  A process is a set of executable instructions (program) which runs on a processor. One or more processes may execute on a processor. In a message passing system, all processes communicate with each other by sending messages - even if they are running on the same processor. For reasons of efficiency, however, message passing systems generally associate only one process per processor.
- **Message Passing Library**
  Usually refers to a collection of routines which are imbedded in application code to accomplish send, receive and other message passing operations.
- **Send / Receive**
  Message passing involves the transfer of data from one process (send) to another process (receive). Requires the cooperation of both the sending and receiving process. Send operations usually require the sending process to specify the data's location, size, type and the destination. Receive operations should match a corresponding send operation.

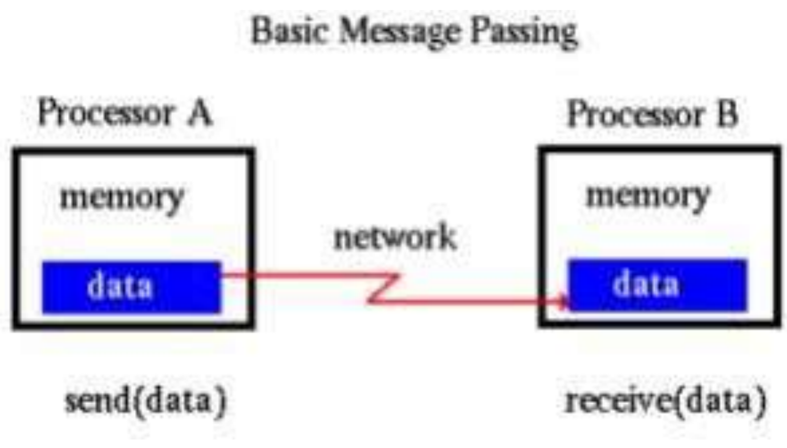


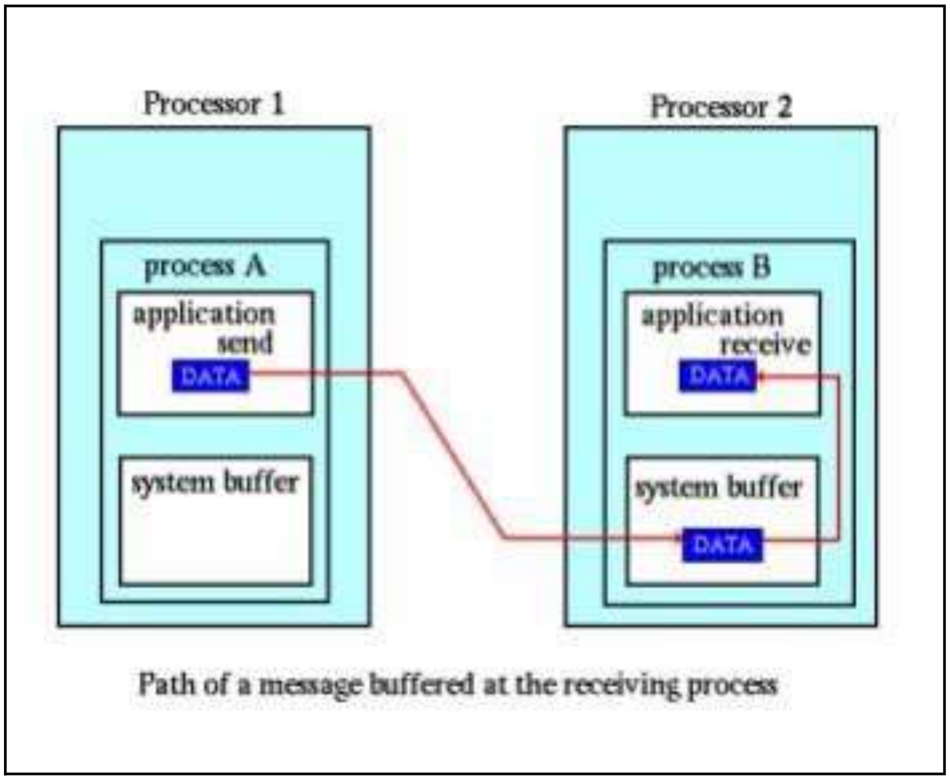
Figure: Send & Receive Msg in MPI in Network

- **Synchronous / Asynchronous**

  A synchronous send operation will complete only after acknowledgement that the message was safely received by the receiving process. Asynchronous send operations may "complete" even though the receiving process has not actually received the message.
- **Application Buffer**

  The address space that holds the data which is to be sent or received. For example, your program uses a variable called, "in msg". The application buffer for in msg is the program memory location where the value of in msg resides.
- **System Buffer**

  System space for storing messages. Depending upon the type of send/ receive operation, data in the application buffer may be required to be copied to/from system buffer space. Allows communication to be asynchronous.



- **Blocking Communication**
  A communication routine is blocking if the completion of the call is dependent on certain "events". For sends, the data must be successfully sent or safely copied to system buffer space so that the application buffer that contained the data is available for reuse. For receives, the data must be safely stored in the receive buffer so that it is ready for use.
  to effect performance gains.

Advantage of MPI

1. Standardization: MPI is the only message passing library which can be considered a standard.
2. Portability: no need to modify your source code when you port your application to a different platform.
3. Functionality: Many routines available to use.
4. Availability: A variety of implementations are available.

Disadvantages of MPI:

1. Lack of scalability between memory and CPU. Adding more CPUs can increase the traffic on shared memory associated with cache and memory management.
2. Programmer responsibility for synchronization construct that ensure correct access of global memory.
3. Expense: It becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing number of processors.

- **Case Study: BSD UNIX IPC Mechanism**
  Feature:

1. It is network independent in the sense that it can support communication networks that use different sets of protocols, different naming conversions, different hardware, and so on.

2. It uses a unified abstraction, called socket, for an endpoint of communication. That is, a socket is an abstract object form which message are sent and received.

3. For location transparency, it uses a two-level naming scheme for naming communication endpoints That is socket can be assign a high-level name that is a human readable string. 4.

It is highly flexible in the sense that it uses a typing mechanism for socket to provide the semantic aspect of communication to application in a controlled and uniform manner. 🎬 **IPC Primitives:**

1. S=socket(domain, type, protocol)

When a process wants to communicate with another process, it must first create a socket by using the socket system call. The first parameter specify the communication domain, second parameter specifies the socket type and third parameter specifies the communication protocol.

2. Bind(s, addr, addrlen)

After creating a socket, the receiver must bind it to a socket address. If two-way communication is desired between two processes, both processes have to receive messages, and hence both must separately bind their sockets to a socket address.

3. Connect(s, server_addr, server_addrlen)

In connection-based communication, two processes first establish a connection between their paires of sockets. The connection establishment pricess is asymmetric because one of the processes keeps waiting for a request for a connection and the other makes a request for a connection.

4. Listen(s, backlog)

The listen system call is used in case of connection-based communication by a server process to listen in its socket for client requests for connections.

5. Snew=accept(s, client_addr, client_addrlen)

The accept system call is used in a connection-based communication by a server process to accept a request for a connection establishment made by a client and to obtain a new socket for a communication with that client.

6. Primitives for Sending and Receiving Data:

nbytes= read(snew, buffer, amount) write(s,
“message”, msg_length) amount=
recvform(s, buffer, sender_address)
sendto(s, “message”, receiver_address)

Conclusion:

In this Practical, we studied about Message Passing Interface

……………………………………

Miss. Archana Chitte

Name & Sign of Course Teacher