SMVS
Page No.
Date : / /

Message Passing Computing

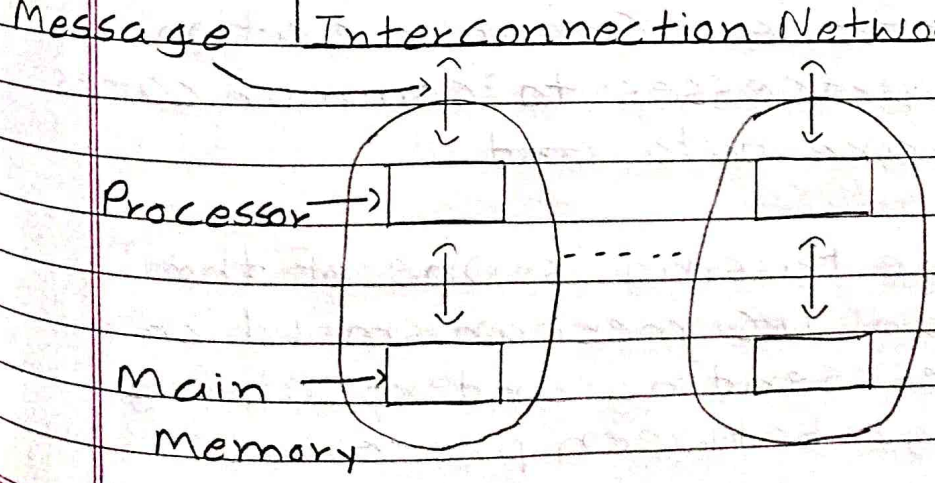
1/ What is Message Passing Computing.
4. Explain Passing a message between processes using send() and recv() Library calls.

=> Message Passing Computing:

Message Passing is a communication technique used in parallel and distributed computing system.

Using Message Passing Computing, Processes communicate by sending and receiving message.

Message Passing involves the information or instruction that need to be communicated from one process to another.



This Process Used Send and Receive Operation.

Send : A Process send a message to another Process

Receive : A Process waits for and retrieves a message sent by another.

-> There are Main Four Types of Message Passing:

(1) Synchronous
(2) Asynchronous
(3) Point-to-Point Communication
(4) Collective Communication

=> Passing Messages Between Processes Using send() and Recv().

Inter-Process Communication allows processes to communicates and share data. ~~End~~

Message Passing is one of the common IPC mechanism which involves sending and receiving message between processes

SMVS
Page No.
Date : / /

using the send( ) and recv( ) Library calls.

When two Processes need to communicate, one process send a message and the other process receives it.

(1) Send( ) Call:

The send( ) Function is used by the source process to send a message to another process.

It usually includes two parameters.
- Data : Data that need to be sent.
- Destination ID : The ID of the Destination process.

Example : send(&data, destination_process id);

The send( ) call may block the sender process until the message is successfully transfered.

The Sender waits until an acknowledgment from the receiver process, which makes Synchronous communication.

(2) recvC) call:

The recvC) function is used by the destination process to receive the message.

It include two parameters.
- Data Variable: A Location where the received data will be stored.
- Source ID: The ID of the Process from which the message is expected.

Example: recv(&received_data, source_process_id);

The recvC) call can also block the receiver process until a message is available.

This ensures that the receiver does not proceed without receiving the expected data.

-> The Type and size of the data sent and received must match between sender and receiver.

SMVS
Page No.
Date : / /

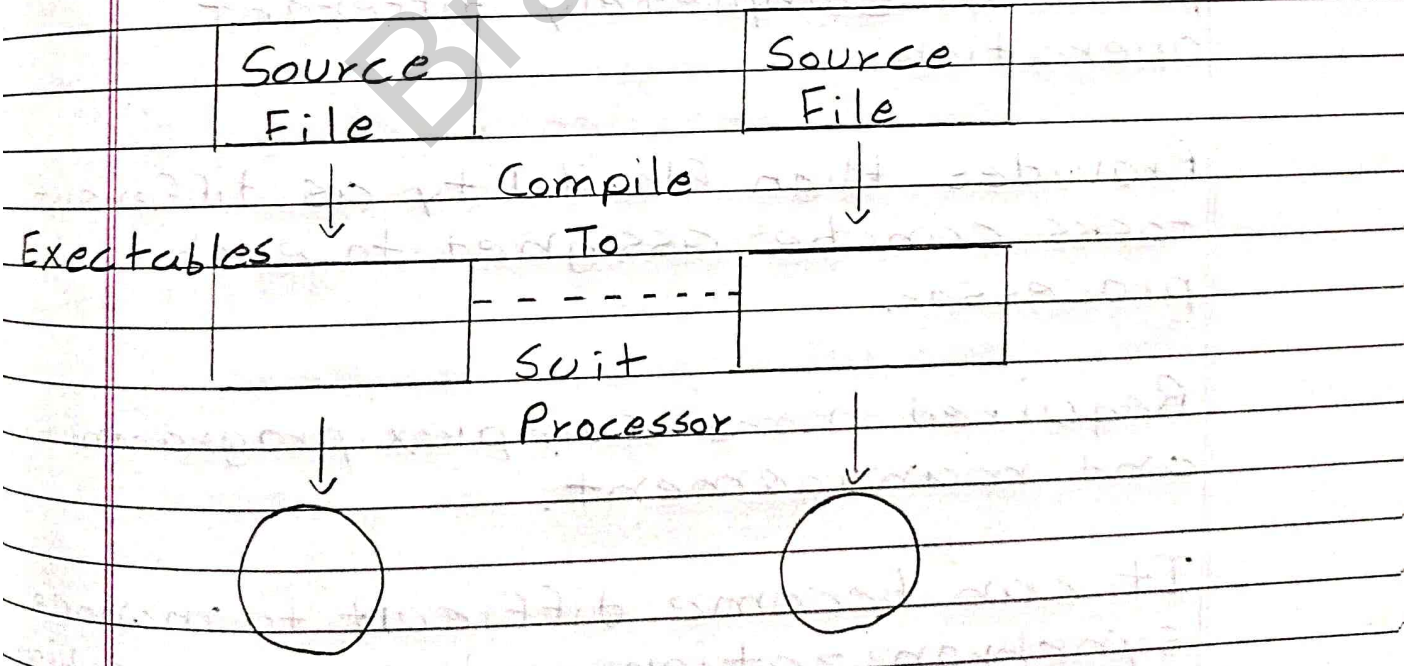
2 Explain Process Creation using both models as MPMD and SPMD model.

=> MPMD Model:

MPMD stands for Multiple-Program, Multiple Data.

In the MPMD Model, each processor run a different program.

There are multiple distinct programs, each excuted by different processors in a parallel computing system.



In MPMD, Each Processor runs a unique program and each program can have different task or functionalities.

Each Processor may execute a different part of the application.

In this model, coordination between the processor is more complex because processor is running a different program.

This model is suitable when different processor need to perform completely different operation.

Provides High flexibility as different tasks can be assigned to each processor.

Required more complex programming and management.

It can become difficult to manage synchronization and communication between different program.

SMVS
Page No.
Date : / /

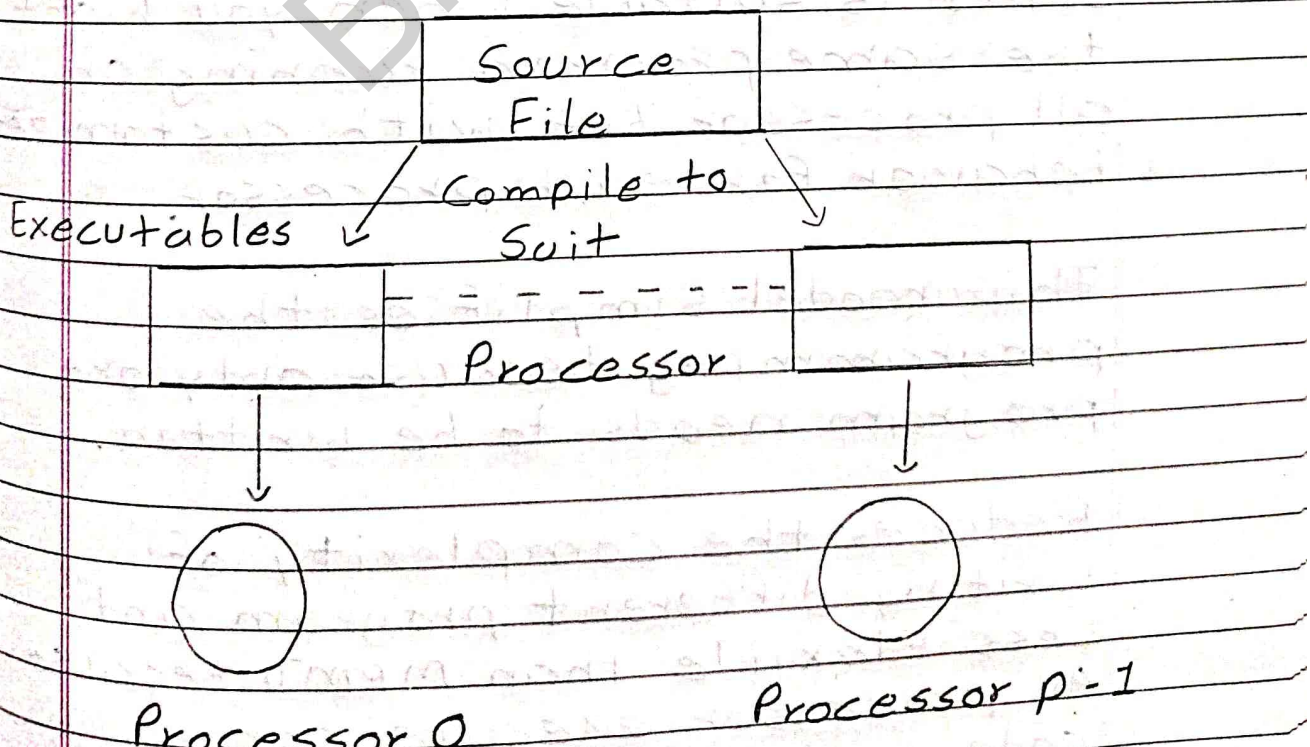
=> SPMD Model :

SPMD stands for Single - Program Multiple - Data.

In the SPMD model, A single Program is executed on all processors.

But the Program can behave differently on each processor, depending on the processor's Identity.

In SPMD, there is only one program, which is compiled into executable code for all processor.



SMVS
Page No.
Date : / /

All Processor load the same program and start execution at the same time.

Even though the same program is executed on all processors but the control flow can differ.

The differentiation occurs based on the process ID (A unique identifier for each processor).

The Program itself contains control structures that differentiates tasks.

SPMD is suitable when you want the same program running on all processor but with customized behavior for each processor.

This model simplifies the programming because only one program needs to be written.

Reduces the complexity of writing different program and Less Flexible than MPMD because All processor execute same code.

SMVS
Page No.
Date : / /

3 Difference between Static Process and Dynamic Process creation.

=>

	Static Process Creation	Dynamic Process Creation
1	All the Processes are created before execution.	All the Processes are created during execution.
2	Fixed number of processes throughout execution.	Number of Processes can change during execution.
3	All Processes are explicitly defined in advance.	Processes are created on-demand by the program.
4	Less Scalable, as the number of process is Fixed.	More scalable, as process can be add/remove as needed.
5	Simpler to implement and manage.	More Complex due to runtime management.
6	Lower Overhead	Higher Overhead

Date : / /

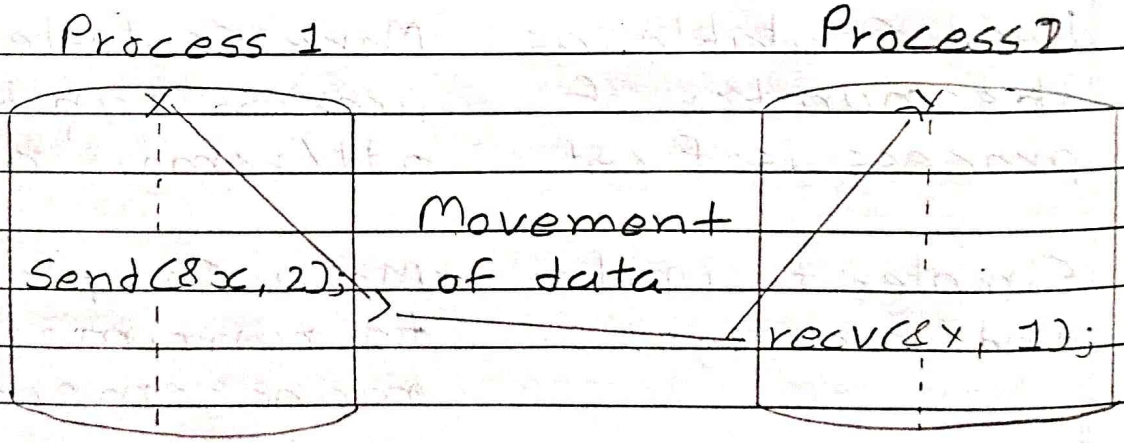
5 Explain the following with their suitable diagrams.

A Synchronous and Asynchronous Message Passing.

=> Synchronous Message Passing:

Synchronous implies synchronization between two processes. Both processes must be synchronized at the point of communication before they can proceed with work.

In Synchronous message passing, the send() and recv() calls are blocking.



(Passing Message)

The sender blocks until the message is delivered and The

receiver blocks until the message is available.

This is typically done using a three way handshake method.

1 -> The sender sends a "request to send" message to the receiver.

2 -> The receiver sends an acknowledgment that its ready to accept message.

3 -> Sender then sends the actual data and the receiver receives it.

=> Asynchronous Message Passing:

In Asynchronous Message Passing the sender doesn't wait for the message to the received before continuing its execution.

The receiver may also check periodically or be notified once the message is received.

Date : / /

Message Passing Library calls send() and recv() do not block.

This approach is used for Low-latency communication and acheived non-blocking behavior.

B Blocking and NonBlocking Message Passing

=> Blocking Message Passing:

It refers to a communication operation where the process that initiates the message transfer waits until the transfer waits until the transfer is complete before continuing its execution.

The Primary feature of blocking message-passing is that it synchronizes the sender and receiver.

In blocking communication, both the send and receiver operations block the process.

SMVS
Page No.
Date : / /

-> Example:

Process 1: Calls send() to send data

Process 2: Calls recv() to receive the data.

Process 1 is blocked until Process 2 calls recv(), and Process 2 is blocked until Process 1 calls send()

=> Non Blocking Message Passing:

Nonblocking message passing allows a process to initiate a message transfer and continue executing other tasks without waiting for the transfer to complete.

Nonblocking operations allow processes to do other work while waiting for message to be received.

The sender does not wait for an acknowledgment from the receiver that the message has been received.

SMVS
Page No.
Date : / /

-> Example:

Process 1 calls send() to send data.

The send operation immediately returns, even if the message is still in transit.

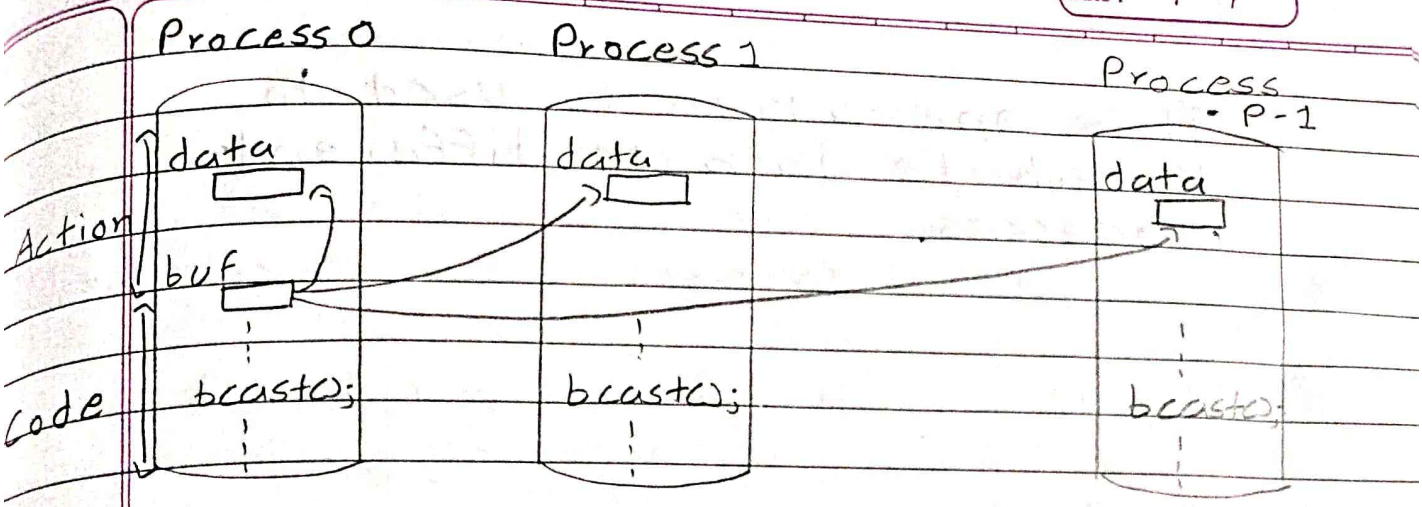
Process 2 calls recv() and if the message is not yet available, it can continue doing other work.

6 Write a short note on the following operations.

(A) Broadcast Operation:

Broadcast is the process of sending the same message or data to all participating process.

It is used in scenarios where all processes involved in a task and need to receive the same information.



In the Process, Root Process is identified and root process sends the same message to all processes that need to receive it.

Every Process involved in the operation executes the same bcast() routine.

The Broadcast operation ensures that the processes synchronize at the point of communication.

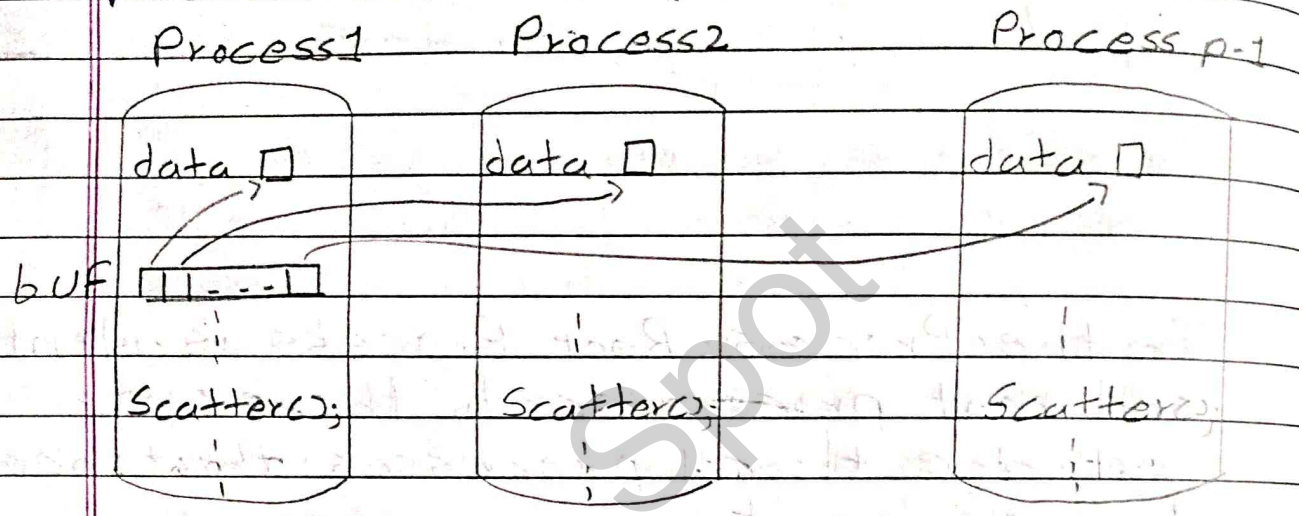
(B) Scatter Operation:

Scatter involves sending different parts of an array or data set to different processes.

Each element of an array in the root process is distributed to a corresponding process.

SMVS
Page No.
Date : / /

This operation is used to distribute data to different processes.



The root process ~~hols~~ holds an array or collection of data, this data is divided into chunks and each chunk is sent to specific process.

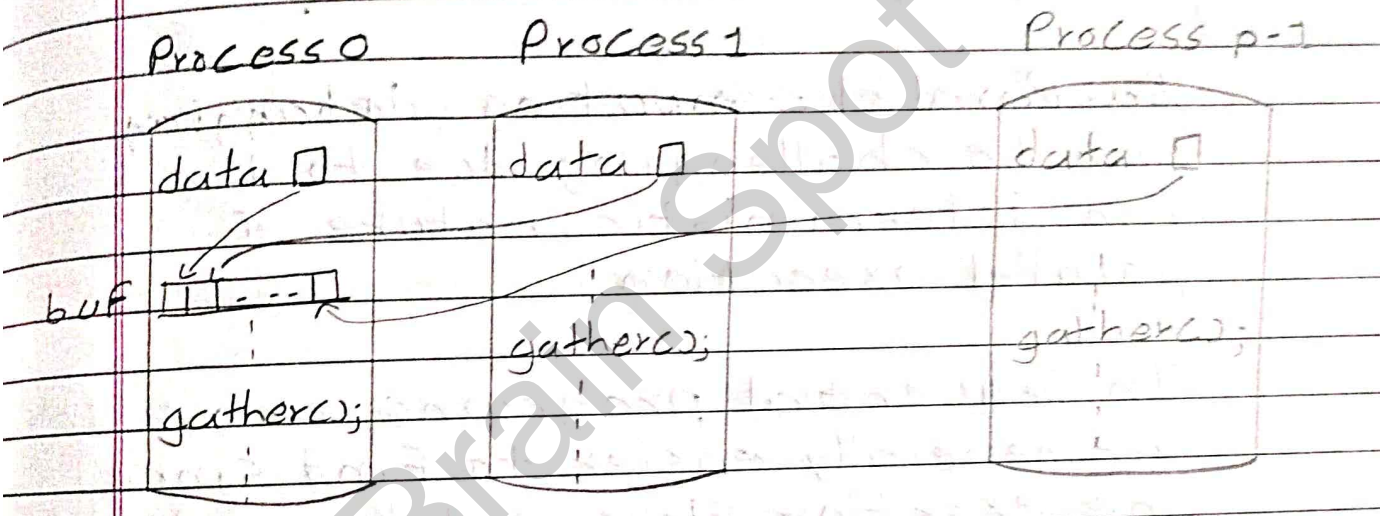
The scatter operation is used in parallel algorithms to divide the workload.

Each process perform the same scatter() routine to receive its portion of data.

(c) Gather Operation:

Gather is the inverse of scatter, it is the process where one process collects data from multiple processes.

Each process sends its result or data back to the root process.



The root process collects these values and store them in an array or another data structure.

The gather operation can be used in parallel algorithms to aggregate result after the parallel computation is done.

Collects data from multiple processes into the one process.

Page No.
Date : / /

7/ 8 What is Low level debugging. Explain debugging and Evaluating parallel programs empirically.

=> Low Level Debugging:

It refers to debugging techniques that involves a detailed examination of a program's execution.

In Parallel computing, debugging can be challenging due to the non-deterministic nature of parallel execution.

In Sequential programs, errors are generally easier to find since the exection flow is deterministic.

=> Debugging:

Geist et al. propose a three step approach to debugging message-passing parallel programs.

(1) Run as a Single Process:

Start by debugging the program in a sequential form to ensure that the basic Logic works.

SMVS
Page No.
Date : / /

(2) Multitask on One Computer:

Use 2-4 Processes on a single machine to check for synchronization errors.

(3) Test across Multiple Computer:

Run the program on several machines to identify network-related issues.

=> Evaluating Programs:

(1) Measuring Execution Time:

Only when the algorithm is coded and executed on a multiprocessor system will give the actual performance of algorithm.

We can use system calls to measure execution time such as time() or gettimeofday().

(2) Communication by the Ping-Pong Method:

Measures Point-to-Point communication

time between two process.

It is used to identify network latency or delays and benchmark the performance of inter-process communication.

Process:

(i) Process Po sends a message to P1.

(ii) As soon as P1 receives the message, it sends it back to Po.

(iii) The time taken for this round trip is measured at Po, and the total time is divided by 2 to estimate one-way communication time.

(3) Profiling:

Analyzes the performance of a program by recording how much time is spent in different parts of the program.

SMVS
Page No.
Date : / /

It is used to understand where time is spent in program.

Use statistical sampling to collect data on program execution.

Used to identify parts of the program that consume the most resources.

Profiling itself can alter the program's behaviour or execution time due to its invasive nature.