Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA: 3.18)

JUNAID GIRKAR | 60004190057 | BE COMPS A2 | HPC | EXP 2

EXPERIMENT 2

AIM: To implement the following:

1. Program to send data and receive data to/from processors using MPI
2. Program illustrating broadcast of data using MPI

THEORY:

Sending and receiving are the two foundational concepts of MPI. Almost every single function in MPI can be implemented with basic send and receive calls.

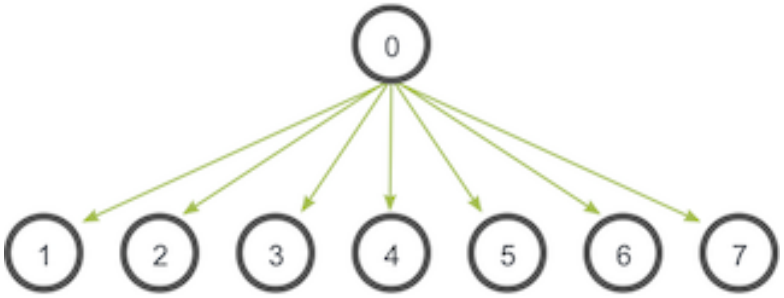
MPI's send and receive calls operate in the following manner. First, process A decides a message needs to be sent to process B. Process A then packs up all of its necessary data into a buffer for process B. These buffers are often referred to as envelopes since the data is being packed into a single message before transmission (similar to how letters are packed into envelopes before transmission to the post office). After the data is packed into a buffer, the communication device (which is often a network) is responsible for routing the message to the proper location. The location of the message is defined by the process's rank.

Even though the message is routed to B, process B still has to acknowledge that it wants to receive A's data. Once it does this, the data has been transmitted. Process A is acknowledged that the data has been transmitted and may go back to work.

Sometimes there are cases when A might have to send many different types of messages to B. Instead of B having to go through extra measures to differentiate all these messages, MPI allows senders and receivers to also specify message IDs with the message (known as tags). When process B only requests a message with a certain tag number, messages with different tags will be buffered by the network until B is ready for them.

A broadcast is one of the standard collective communication techniques. During a broadcast, one process sends the same data to all processes in a communicator. One of the main uses of broadcasting is to send out user input to a parallel program, or send out configuration parameters to all processes.

The communication pattern of a broadcast looks like this:



In this example, process zero is the root process, and it has the initial copy of data. All of the other processes receive the copy of data.

1. **CODE:**

```
from mpi4py import MPI

def main():
  passes = 5
  id = ""
  rank = MPI.COMM_WORLD.Get_rank()
  comm = MPI.COMM_WORLD
  if rank == 0:
        id = "Player 0"
        next = 1
  else:
        id = "Player 1"
        next = 0

  if rank == 0:
        print(f"Number of Parallel Players is {comm.Get_rank()}")
        comm.send("Player 0 Initiates", dest = 1, tag = 0)

  while passes > 0:
        msg = comm.recv(source = next, tag = 0)
        print(f"{id} received : {msg}")
```

```
        comm.send(f"{id}'s {5 - passes + 1}th pass", dest = next, tag = 0)
        passes -= 1

 print("Passes over for ", id)

if __name__ == "__main__":
 main()
```

OUTPUT:

```
jarvis@jarvis-Inspiron-7591:~/Desktop$ mpiexec -n 4 python3 sendreceive.py
Number of Parallel Players is 0
Player 1 received : Player 0 Initiates
Player 1 received : Player 0's 1th pass
Player 0 received : Player 1's 1th pass
Player 0 received : Player 1's 2th pass
Player 0 received : Player 1's 3th pass
Player 1 received : Player 0's 2th pass
Player 1 received : Player 0's 3th pass
Player 0 received : Player 1's 4th pass
Player 1 received : Player 0's 4th pass
Player 0 received : Player 1's 5th pass
Passes over for  Player 0
Passes over for  Player 1
```

---

2. **CODE:**

master.py

```
from mpi4py import MPI
import random
random.seed(random.random())
```

```
def master(rank):
  steps = random.randint(1,10)
  print("Master Steps : ", steps)
  for i in range(steps):
    print(f"Master rank {rank} working for {i}th time")

def main():
  rank = MPI.COMM_WORLD.Get_rank()
  comm = MPI.COMM_WORLD
  master(rank)
  comm.Barrier()
  comm.bcast("Mission Successful", root = rank)

if __name__ == "__main__":
  main()
```

slave.py

```
from mpi4py import MPI
import random
random.seed(random.random())

def slave(rank):
  steps = random.randint(1,10)
  print("Slave Steps : ", steps)
  for i in range(steps):
    print(f"Slave rank {rank} working for {i}th time")

def main():
  rank = MPI.COMM_WORLD.Get_rank()
  comm = MPI.COMM_WORLD
  slave(rank)
  data = ""
  comm.Barrier()
  data = comm.bcast(data, root = 0)
  print(f"Slave Rank {rank} receives: {data}")
```

```
if __name__ == "__main__":
  main()
```

OUTPUT:

```
D:\SEM 7\HPC\EXPERIMENT 2>mpiexec /np 1 python master.py : /np 3 python
slave.py
Master Steps :  8
Master rank 0 working for 0th time
Master rank 0 working for 1th time
Master rank 0 working for 2th time
Master rank 0 working for 3th time
Master rank 0 working for 4th time
Master rank 0 working for 5th time
Master rank 0 working for 6th time
Master rank 0 working for 7th time
Slave Steps :  7
Slave rank 2 working for 0th time
Slave rank 2 working for 1th time
Slave rank 2 working for 2th time
Slave rank 2 working for 3th time
Slave rank 2 working for 4th time
Slave rank 2 working for 5th time
Slave rank 2 working for 6th time
Slave Rank 2 receives: Mission Successful
Slave Steps :  1
Slave rank 3 working for 0th time
Slave Rank 3 receives: Mission Successful
Slave Steps :  6
Slave rank 1 working for 0th time
Slave rank 1 working for 1th time
Slave rank 1 working for 2th time
Slave rank 1 working for 3th time
Slave rank 1 working for 4th time
Slave rank 1 working for 5th time
```

```
Slave Rank 1 receives: Mission Successful
```

CONCLUSION: Sending and receiving messages is an integral part of MPI. In this experiment, we have demonstrated a program that sends and receives data to/from processors. Along with this, we have also demonstrated data broadcast using MPI in Python.