Computer Science & Engineering

CSE4001

Parallel and Distributed Computing

LAB ASSIGNMENT 7

Submitted to **Prof. DEEBAK B.D.**

TOPIC: PROBLEMS USING MPI

NAME: PUNIT MIDDHA

REG.NO: 19BCE2060

SLOT: L55+L56

DATE: 10/11/2021

QUESTION – I

Consider the following program, called mpi_sample1.c. This program is written in C with MPI commands included.

The new MPI calls are to MPI_Send and MPI_Recv and to MPI_Get_processor_name. The latter is a convenient way to get the name of the processor on which a process is running. MPI_Send and MPI_Recv can be understood by stepping back and considering the two requirements that must be satisfied to communicate data between two processes:

1. Describe the data to be sent or the location in which to receive the data
2. Describe the destination (for a send) or the source (for a receive) of the data.

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char* argv[]){
	int my_rank; /* rank of process */
	int p; /* number of processes */
	int source; /* rank of sender */
	int dest; /* rank of receiver */
	int tag=0; /* tag for messages */
	char message[100]; /* storage for message */
	MPI_Status status ; /* return status for receive */
	/* start up MPI */
	[redacted]
	/* find out process rank */
	[redacted]
	/* find out number of processes */
	[redacted]
		/* create message */
		[redacted]
		/* use strlen+1 so that '\0' get transmitted */
		[redacted]
	}
	else{
		[redacted]
		}
	}

	/* shut down MPI */
	[redacted]
	return 0;
}
```

1. Implement the above code
2. Build and Execute the logical scenario with few test cases
3. Depict the screenshots along with proper justification

SOURCE CODE:

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char* argv[]){
int my_rank; /* rank of process */
int p; /* number of processes */
int source; /* rank of sender */
int dest; /* rank of receiver */
int tag=0; /* tag for messages */
char message[100];

/* storage for message */
MPI_Status status ; /* return status for receive */

/* start up MPI */
MPI_Init(&argc, &argv);

/* find out process rank */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

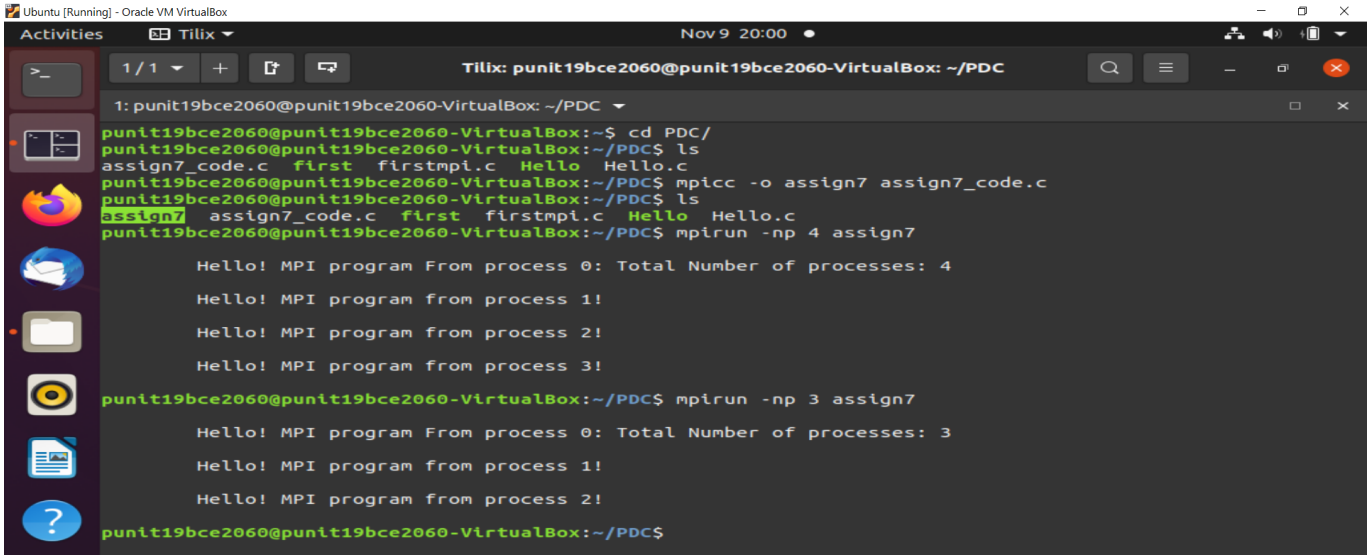
```
/* find out number of processes */
MPI_Comm_size(MPI_COMM_WORLD, &p);

if (my_rank !=0){
/* create message */
sprintf(message, "\tHello! MPI program from process %d!\n", my_rank);
dest = 0;
/* use strlen+1 so that '\0' get transmitted */
MPI_Send(message, strlen(message)+1, MPI_CHAR,
dest, tag, MPI_COMM_WORLD);
}

else{
printf("\n\tHello! MPI program From process 0: Total Number of processes: %d\n\n",p);
for (source = 1; source < p; source++) {
MPI_Recv(message, 100, MPI_CHAR, source, tag,
MPI_COMM_WORLD, &status);
printf("%s\n",message);
}
}

/* shut down MPI */
MPI_Finalize();
return 0;
}
```

EXECUTION:



REMARKS:

- ✓ MPI_INIT is used to initiate an MPI computation with necessarily two arguments as &argc, &argv.
- ✓ MPI_COMM_SIZE: This function is used to count the number of processes.
- ✓ MPI_COMM_RANK: This function is used to find my process identifier.
- ✓ MPI_COMM_WORLD: MPI COMM WORLD function is used to communicates. This is basically called as communicator.
- ✓ MPI_Send, to send a message to another process.
- ✓ MPI_Recv, to receive a message from another process.
- ✓ MPI_FINALIZE: This function is used to ends a computation also it is used at the end of code.
- ✓ It is clearly visible that data send and receive as mentioned is the question.