



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

## **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]
    [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);
```

```

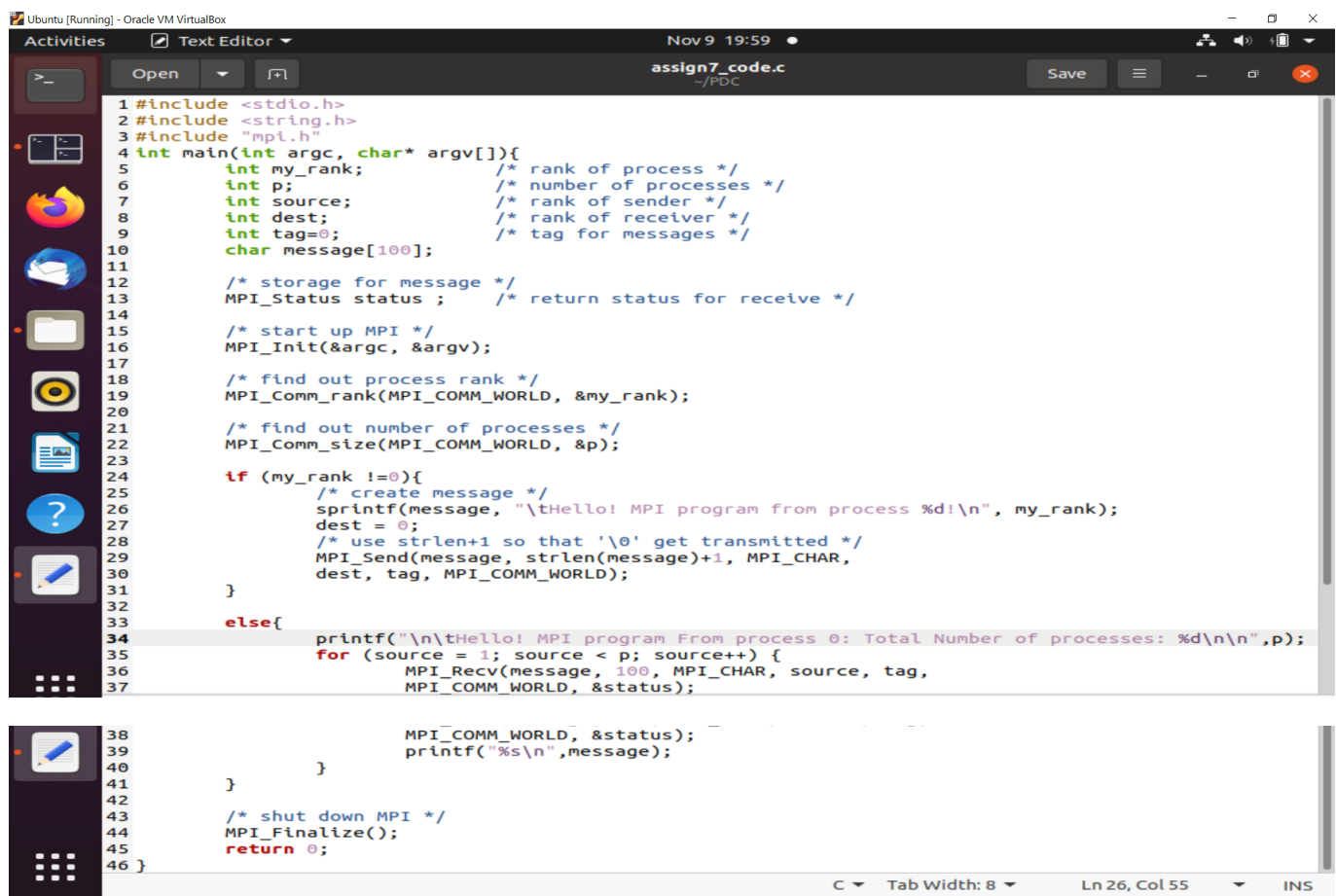
/* find out number of processes */
MPI_Comm_size(MPI_COMM_WORLD, &p);

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;
}

```

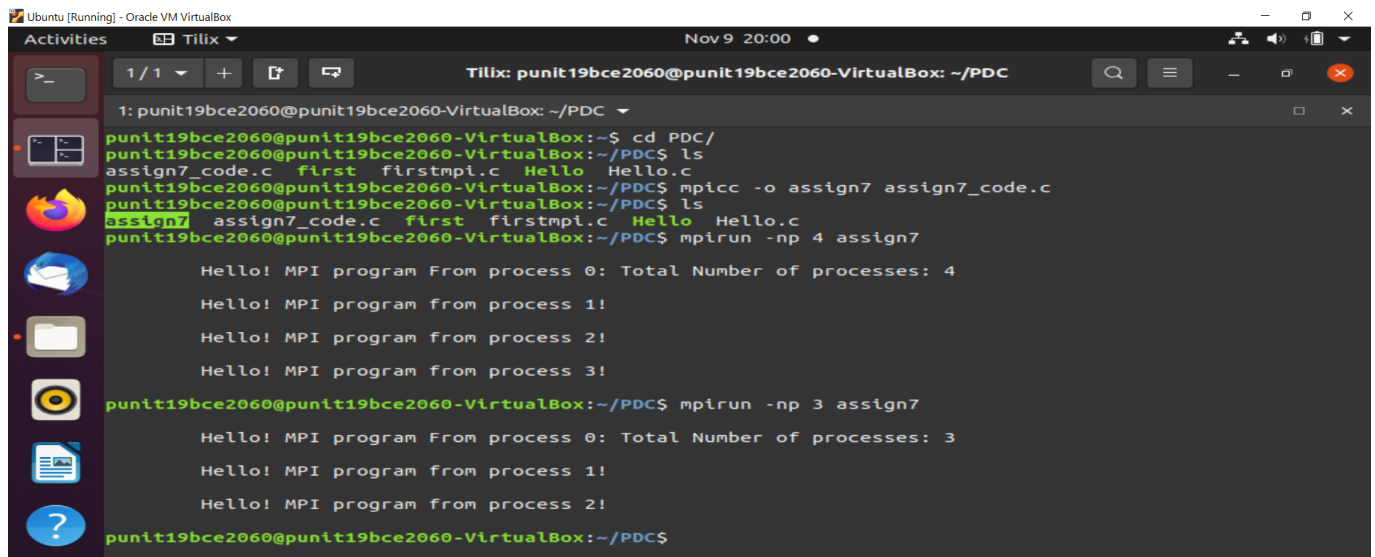


```

1 #include <stdio.h>
2 #include <string.h>
3 #include "mpi.h"
4 int main(int argc, char* argv[]){
5     int my_rank;           /* rank of process */
6     int p;                 /* number of processes */
7     int source;            /* rank of sender */
8     int dest;              /* rank of receiver */
9     int tag=0;             /* tag for messages */
10    char message[100];
11
12    /* storage for message */
13    MPI_Status status;      /* return status for receive */
14
15    /* start up MPI */
16    MPI_Init(&argc, &argv);
17
18    /* find out process rank */
19    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
20
21    /* find out number of processes */
22    MPI_Comm_size(MPI_COMM_WORLD, &p);
23
24    if (my_rank !=0){
25        /* create message */
26        sprintf(message, "\tHello! MPI program from process %d!\n", my_rank);
27        dest = 0;
28        /* use strlen+1 so that '\0' get transmitted */
29        MPI_Send(message, strlen(message)+1, MPI_CHAR,
30        dest, tag, MPI_COMM_WORLD);
31    }
32
33    else{
34        printf("\n\tHello! MPI program From process 0: Total Number of processes: %d\n\n",p);
35        for (source = 1; source < p; source++) {
36            MPI_Recv(message, 100, MPI_CHAR, source, tag,
37            MPI_COMM_WORLD, &status);
38            printf("%s\n",message);
39        }
40    }
41
42    /* shut down MPI */
43    MPI_Finalize();
44    return 0;
45 }
46

```

## **EXECUTION:**



```
punit19bce2060@punit19bce2060-VirtualBox:~$ cd PDC/
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign7_code.c  first  firstmpi.c  Hello  Hello.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpicc -o assign7 assign7_code.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign7  assign7_code.c  first  firstmpi.c  Hello  Hello.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpirun -np 4 assign7

Hello! MPI program From process 0: Total Number of processes: 4
Hello! MPI program from process 1!
Hello! MPI program from process 2!
Hello! MPI program from process 3!

punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpirun -np 3 assign7

Hello! MPI program From process 0: Total Number of processes: 3
Hello! MPI program from process 1!
Hello! MPI program from process 2!

punit19bce2060@punit19bce2060-VirtualBox:~/PDC$
```

## **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.