



VIT[®]
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 10

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

TOPIC: PROBLEMS USING MPI

NAME: PUNIT MIDDHA

REG.NO: 19BCE2060

SLOT: L55+L56

DATE: 03/12/2021

QUESTION – I

Assume the variable rank contains the process rank and root is 3. What will be stored in array b [] on each of four processes if each executes the following code fragment?

```
int b [4] = {0 , 0 , 0 , 0};
```

```
MPI_Gather ( &rank , 1 , MPI_INT , b , 1 , MPI_INT , root ,MPI_COMM_WORLD);
```

Hint. The function prototype is as follows:

```
int MPI_Gather (  
void * sendbuf ,           // pointer to send buffer  
  
int sendcount ,           // number of items to send  
  
MPI_Datatype sendtype ,   // type of send buffer data  
  
void * recvbuf ,          // pointer to receive buffer  
  
int recvcnt ,             // items to receive per process  
  
MPI_Datatype recvtype ,   // type of receive buffer data  
  
int root ,                // rank of receiving process  
  
MPI_Comm comm )          // MPI communicator to use
```

SOURCE CODE:

```
#include <mpi.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(int argc, char** argv) {  
  
    //Starting MPI  
    MPI_Init(&argc, &argv);  
  
    //Size of processes  
    int size;  
    MPI_Comm_size(MPI_COMM_WORLD, &size);  
  
    //Array initialization  
    int b[4] = {0,0,0,0};  
    if(size != 4)  
    {  
        printf("Minimum 4 MPI processes required.\n");  
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);  
    }  
    //Root's rank  
    int root = 3;
```

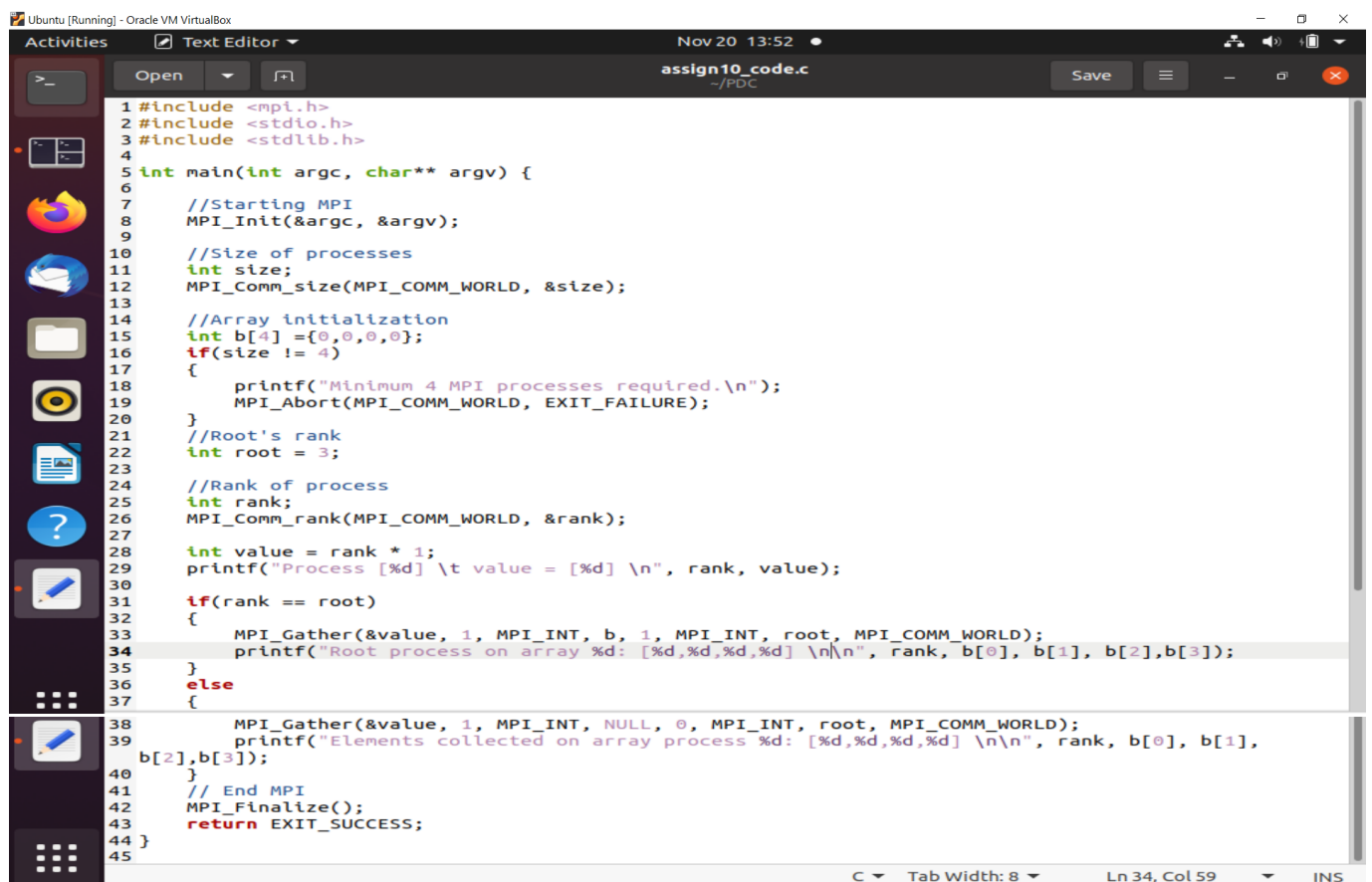
```

//Rank of process
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int value = rank * 1;
printf("Process [%d] \t value = [%d] \n", rank, value);

if(rank == root)
{
    MPI_Gather(&value, 1, MPI_INT, b, 1, MPI_INT, root, MPI_COMM_WORLD);
    printf("Root process on array %d: [%d,%d,%d,%d] \n\n", rank, b[0],
b[1], b[2],b[3]);
}
else
{
    MPI_Gather(&value, 1, MPI_INT, NULL, 0, MPI_INT, root,
MPI_COMM_WORLD);
    printf("Elements collected on array process %d: [%d,%d,%d,%d] \n\n",
rank, b[0], b[1], b[2],b[3]);
}
// End MPI
MPI_Finalize();
return EXIT_SUCCESS;
}

```

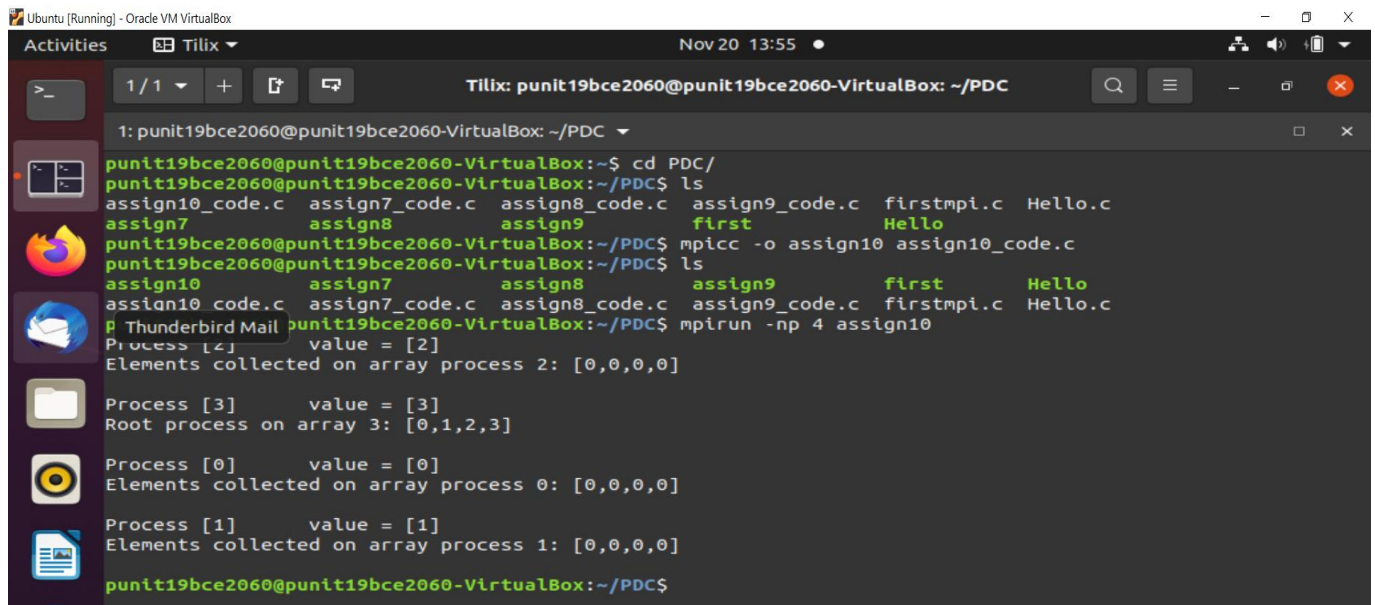


```

1 #include <mpi.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main(int argc, char** argv) {
6
7     //Starting MPI
8     MPI_Init(&argc, &argv);
9
10    //Size of processes
11    int size;
12    MPI_Comm_size(MPI_COMM_WORLD, &size);
13
14    //Array initialization
15    int b[4] = {0,0,0,0};
16    if(size != 4)
17    {
18        printf("Minimum 4 MPI processes required.\n");
19        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
20    }
21    //Root's rank
22    int root = 3;
23
24    //Rank of process
25    int rank;
26    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
27
28    int value = rank * 1;
29    printf("Process [%d] \t value = [%d] \n", rank, value);
30
31    if(rank == root)
32    {
33        MPI_Gather(&value, 1, MPI_INT, b, 1, MPI_INT, root, MPI_COMM_WORLD);
34        printf("Root process on array %d: [%d,%d,%d,%d] \n\n", rank, b[0], b[1], b[2],b[3]);
35    }
36    else
37    {
38        MPI_Gather(&value, 1, MPI_INT, NULL, 0, MPI_INT, root, MPI_COMM_WORLD);
39        printf("Elements collected on array process %d: [%d,%d,%d,%d] \n\n", rank, b[0], b[1],
b[2],b[3]);
40    }
41    // End MPI
42    MPI_Finalize();
43    return EXIT_SUCCESS;
44 }
45

```

EXECUTION:



The screenshot shows a terminal window titled 'Tilix: punit19bce2060@punit19bce2060-VirtualBox: ~/PDC'. The user is in a directory named 'PDC' and lists files: 'assign10_code.c', 'assign7_code.c', 'assign8_code.c', 'assign9_code.c', 'firstmpi.c', and 'Hello.c'. They compile 'assign10_code.c' with 'mpicc -o assign10'. Then they run 'mpirun -np 4 assign10'. The output shows four processes (0, 1, 2, 3) each with a 'value' and 'Elements collected on array process'. Process 3 is the root. The output for each process is: Process [0] value = [0], Elements collected on array process 0: [0,0,0,0]; Process [1] value = [1], Elements collected on array process 1: [0,0,0,0]; Process [2] value = [2], Elements collected on array process 2: [0,0,0,0]; Process [3] value = [3], Root process on array 3: [0,1,2,3].

```
punit19bce2060@punit19bce2060-VirtualBox:~$ cd PDC/
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign10_code.c  assign7_code.c  assign8_code.c  assign9_code.c  firstmpi.c  Hello.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpicc -o assign10 assign10_code.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ ls
assign10         assign7         assign8         assign9         first          Hello
assign10_code.c  assign7_code.c  assign8_code.c  assign9_code.c  firstmpi.c     Hello.c
punit19bce2060@punit19bce2060-VirtualBox:~/PDC$ mpirun -np 4 assign10
Process [0]      value = [0]
Elements collected on array process 0: [0,0,0,0]

Process [1]      value = [1]
Elements collected on array process 1: [0,0,0,0]

Process [2]      value = [2]
Elements collected on array process 2: [0,0,0,0]

Process [3]      value = [3]
Root process on array 3: [0,1,2,3]
```

REMARKS:

- ✓ `b [0, 0, 0, 0]` is an array.
- ✓ The minimal number of processes necessary for the execution of program is equal to the number of array items, which is 4, in this case.
- ✓ The software will provide an error if the number of processes is fewer than 4.
- ✓ Process 3 is assigned as the root.
- ✓ We may use the MPI Gather function to find the array items stored in the supplied array on each of four processes.
- ✓ It Assembles data from all members of a group and distributes it to a single individual.
- ✓ As a result of this, we can see that the array items are shown on all processes except the root one. The process numbers are presented in the root process.