

CSE4001 - Parallel and Distributed Computing

Lab 21+22

Lab Assignment- 6

Submitted by: Alokam Nikhitha

Reg No:19BCE2555

QUESTION 1:

Write a C program to handle message passing in the MPI application interface, which allows processes to communicate with one another. Create two processes that will pass the number 20 from one to the other.

CODE:

```
#include <stdio.h>

#include <mpi.h>

int main(int argc, char** argv) {

    int process_Rank, size_Of_Cluster, message_Item;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);

    MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);

    if(process_Rank == 0){

        message_Item = 20;

        MPI_Send(&message_Item, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);

        printf("Message Sent from Process 0: %d\n", message_Item);

    }

    else if(process_Rank == 1){

        MPI_Recv(&message_Item, 1, MPI_INT, 0, 1, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);

        printf("Message Received in Process 1: %d\n", message_Item);

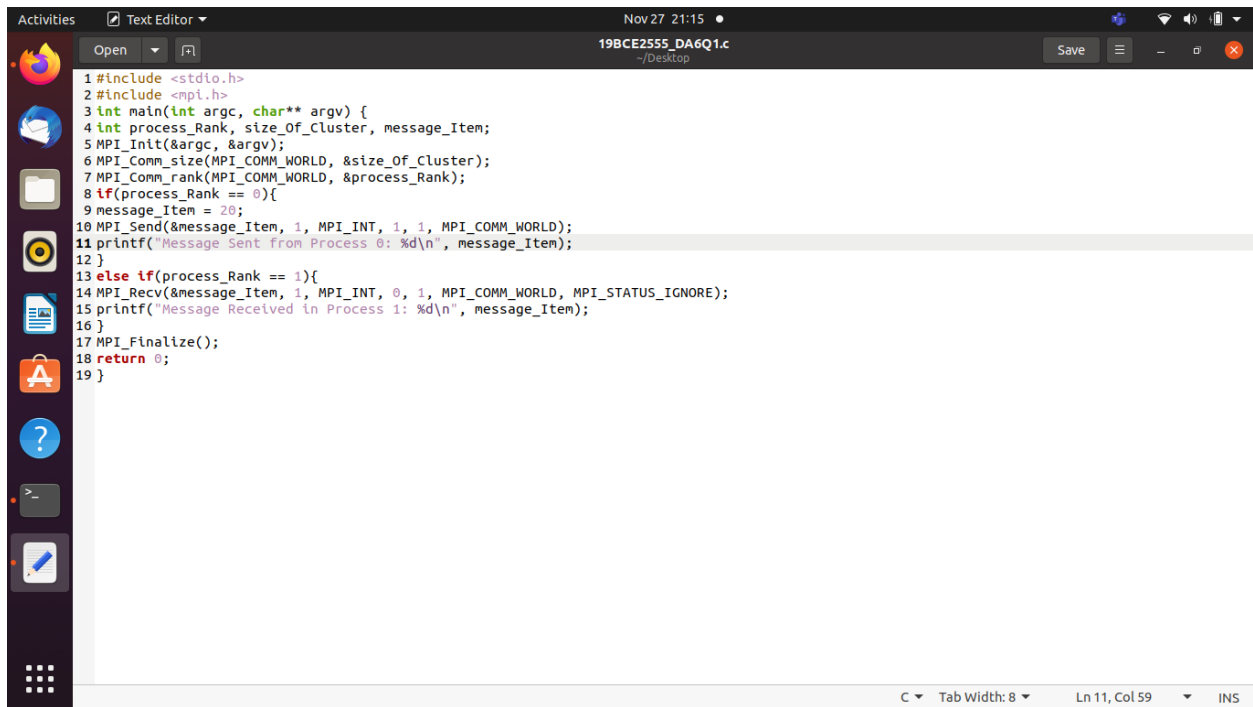
    }

    MPI_Finalize();

    return 0;

}
```

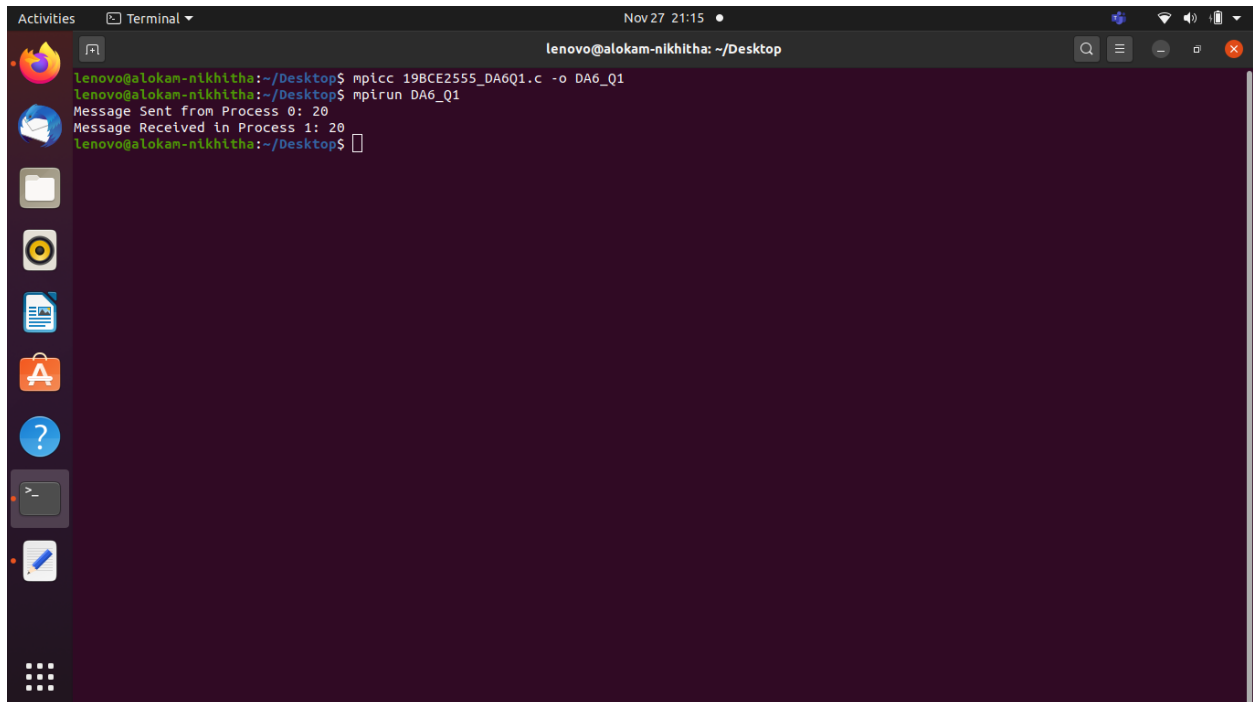
CODE SNIPPETS:



The screenshot shows a text editor window titled "19BCE2555_DA6Q1.c" with the following C code:

```
1#include <stdio.h>
2#include <mpi.h>
3int main(int argc, char** argv) {
4    int process_Rank, size_Of_Cluster, message_Item;
5    MPI_Init(&argc, &argv);
6    MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);
7    MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);
8    if(process_Rank == 0){
9        message_Item = 20;
10       MPI_Send(&message_Item, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
11       printf("Message Sent from Process 0: %d\n", message_Item);
12   }
13   else if(process_Rank == 1){
14       MPI_Recv(&message_Item, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
15       printf("Message Received in Process 1: %d\n", message_Item);
16   }
17   MPI_Finalize();
18   return 0;
19 }
```

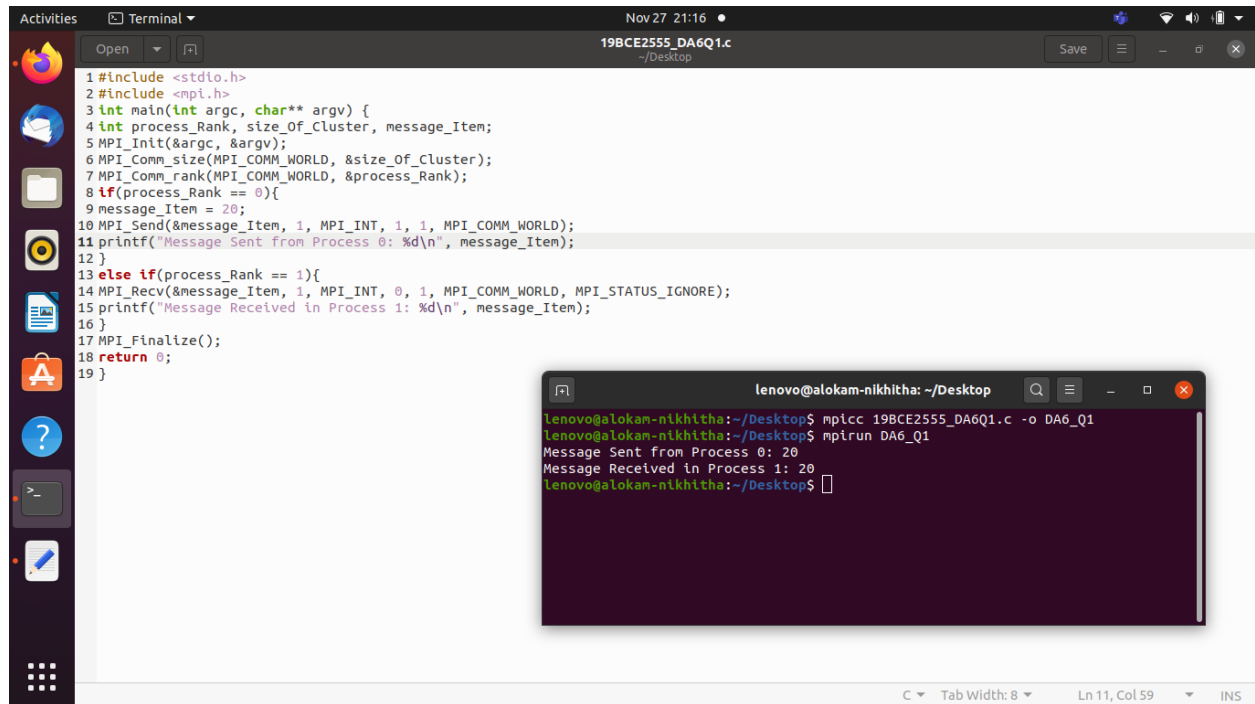
OUTPUT:



The screenshot shows a terminal window with the following output:

```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ mpicc 19BCE2555_DA6Q1.c -o DA6_Q1
lenovo@alokam-nikhitha:~/Desktop$ mpirun DA6_Q1
Message Sent from Process 0: 20
Message Received in Process 1: 20
lenovo@alokam-nikhitha:~/Desktop$
```

OUTPUT WITH CODE:



The screenshot displays a Linux desktop environment with a terminal window open. The terminal shows the source code for a program named `19BCE2555_DA6Q1.c`, which uses MPI to send and receive a message between two processes. The code is as follows:

```
1 #include <stdio.h>
2 #include <mpi.h>
3 int main(int argc, char** argv) {
4     int process_Rank, size_Of_Cluster, message_Item;
5     MPI_Init(&argc, &argv);
6     MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);
7     MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);
8     if(process_Rank == 0){
9         message_Item = 20;
10        MPI_Send(&message_Item, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
11        printf("Message Sent from Process 0: %d\n", message_Item);
12    }
13    else if(process_Rank == 1){
14        MPI_Recv(&message_Item, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
15        printf("Message Received in Process 1: %d\n", message_Item);
16    }
17    MPI_Finalize();
18    return 0;
19 }
```

Below the code, a smaller terminal window shows the execution of the program. The user runs `mpicc 19BCE2555_DA6Q1.c -o DA6_Q1` to compile the code and `mpirun DA6_Q1` to execute it. The output shows the message being sent from Process 0 and received by Process 1, both with the value 20.

```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ mpicc 19BCE2555_DA6Q1.c -o DA6_Q1
lenovo@alokam-nikhitha:~/Desktop$ mpirun DA6_Q1
Message Sent from Process 0: 20
Message Received in Process 1: 20
lenovo@alokam-nikhitha:~/Desktop$
```

Result and Inferences:

- Here we are passing the message between 2 Processes
- We can see that the Number 20 is passed from one Process to other

QUESTION 2:

Write a C program to handle message passing in the MPI application interface using Group Operators: Scatter and Gather.

CODE:

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    int size, rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int *globaldata=NULL;
    int localdata;
    if (rank == 0) {
        globaldata = malloc(size * sizeof(int) );
        for (int i=0; i<size; i++)
            globaldata[i] = 3*i+2;
        printf("Processor %d has data: ", rank);
        for (int i=0; i<size; i++)
            printf("%d ", globaldata[i]);
        printf("\n");
    }
    MPI_Scatter(globaldata, 1, MPI_INT, &localdata, 1, MPI_INT, 0,
MPI_COMM_WORLD);
```

```
printf("Processor %d has data %d\n", rank, localdata);
localdata *= 2;
printf("Processor %d doubling the data, now has %d\n", rank, localdata);

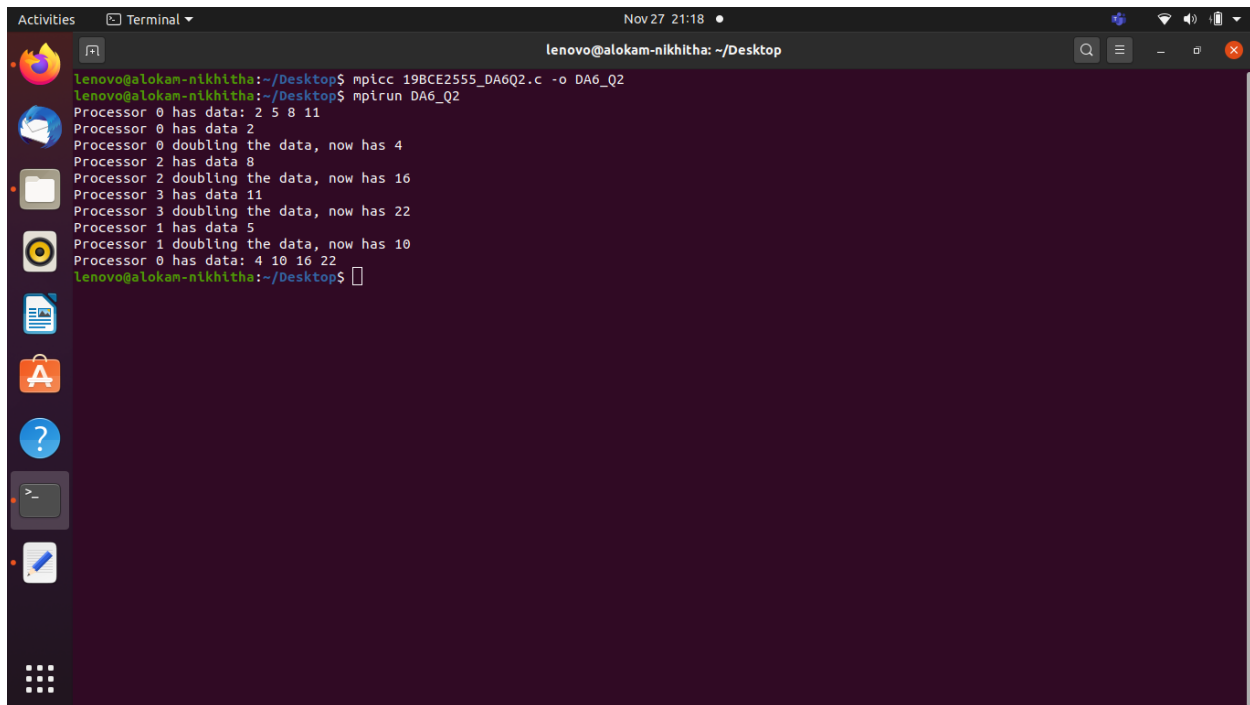
MPI_Gather(&localdata, 1, MPI_INT, globaldata, 1, MPI_INT, 0,
MPI_COMM_WORLD);
if (rank == 0) {
    printf("Processor %d has data: ", rank);
    for (int i=0; i<size; i++)
        printf("%d ", globaldata[i]);
    printf("\n");
}
if (rank == 0)
    free(globaldata);
MPI_Finalize();
return 0;
}
```

Code Snippets:



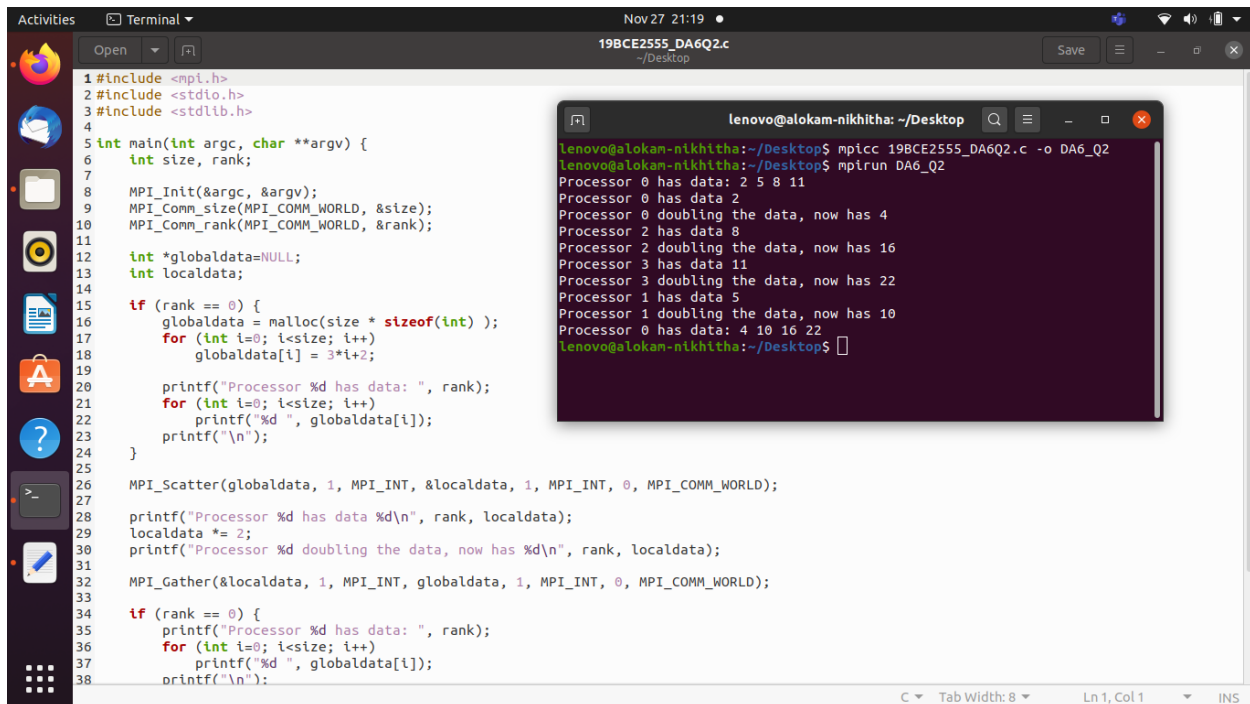
```
1#include <mpi.h>
2#include <stdio.h>
3#include <stdlib.h>
4
5int main(int argc, char **argv) {
6    int size, rank;
7
8    MPI_Init(&argc, &argv);
9    MPI_Comm_size(MPI_COMM_WORLD, &size);
10   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11
12   int *globaldata=NULL;
13   int localdata;
14
15   if (rank == 0) {
16       globaldata = malloc(size * sizeof(int));
17       for (int i=0; i<size; i++)
18           globaldata[i] = 3*i+2;
19
20       printf("Processor %d has data: ", rank);
21       for (int i=0; i<size; i++)
22           printf("%d ", globaldata[i]);
23       printf("\n");
24   }
25
26   MPI_Scatter(globaldata, 1, MPI_INT, &localdata, 1, MPI_INT, 0, MPI_COMM_WORLD);
27
28   printf("Processor %d has data %d\n", rank, localdata);
29   localdata *= 2;
30   printf("Processor %d doubling the data, now has %d\n", rank, localdata);
31
32   MPI_Gather(&localdata, 1, MPI_INT, globaldata, 1, MPI_INT, 0, MPI_COMM_WORLD);
33
34   if (rank == 0) {
35       printf("Processor %d has data: ", rank);
36       for (int i=0; i<size; i++)
37           printf("%d ", globaldata[i]);
38       printf("\n");
39   }
40
41   if (rank == 0)
42       free(globaldata);
43
44   MPI_Finalize();
45   return 0;
46 }
```

OUTPUT:



```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ mpicc 19BCE2555_DA6Q2.c -o DA6_Q2
lenovo@alokam-nikhitha:~/Desktop$ mpirun DA6_Q2
Processor 0 has data: 2 5 8 11
Processor 0 has data 2
Processor 0 doubling the data, now has 4
Processor 2 has data 8
Processor 2 doubling the data, now has 16
Processor 3 has data 11
Processor 3 doubling the data, now has 22
Processor 1 has data 5
Processor 1 doubling the data, now has 10
Processor 0 has data: 4 10 16 22
lenovo@alokam-nikhitha:~/Desktop$
```

OUTPUT WITH CODE:



```
1 #include <mpi.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main(int argc, char **argv) {
6     int size, rank;
7
8     MPI_Init(&argc, &argv);
9     MPI_Comm_size(MPI_COMM_WORLD, &size);
10    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11
12    int *globaldata=NULL;
13    int localdata;
14
15    if (rank == 0) {
16        globaldata = malloc(size * sizeof(int));
17        for (int i=0; i<size; i++)
18            globaldata[i] = 3*i+2;
19
20        printf("Processor %d has data: ", rank);
21        for (int i=0; i<size; i++)
22            printf("%d ", globaldata[i]);
23        printf("\n");
24    }
25
26    MPI_Scatter(globaldata, 1, MPI_INT, &localdata, 1, MPI_INT, 0, MPI_COMM_WORLD);
27
28    printf("Processor %d has data %d\n", rank, localdata);
29    localdata *= 2;
30    printf("Processor %d doubling the data, now has %d\n", rank, localdata);
31
32    MPI_Gather(&localdata, 1, MPI_INT, globaldata, 1, MPI_INT, 0, MPI_COMM_WORLD);
33
34    if (rank == 0) {
35        printf("Processor %d has data: ", rank);
36        for (int i=0; i<size; i++)
37            printf("%d ", globaldata[i]);
38        printf("\n");
39    }
```

```
lenovo@alokam-nikhitha: ~/Desktop
lenovo@alokam-nikhitha:~/Desktop$ mpicc 19BCE2555_DA6Q2.c -o DA6_Q2
lenovo@alokam-nikhitha:~/Desktop$ mpirun DA6_Q2
Processor 0 has data: 2 5 8 11
Processor 0 has data 2
Processor 0 doubling the data, now has 4
Processor 2 has data 8
Processor 2 doubling the data, now has 16
Processor 3 has data 11
Processor 3 doubling the data, now has 22
Processor 1 has data 5
Processor 1 doubling the data, now has 10
Processor 0 has data: 4 10 16 22
lenovo@alokam-nikhitha:~/Desktop$
```


Result and Inferences:

- We are passing the messages using MPI_Scatter and MPI_Gather Commands.
- Initialized the value in array as $3i+2$ where i is the index of the array.
- We doubled its value during using different allocation named ldata.
- We used MPI_Gather command to read data of ldata allocation.
- Finally, We print the values of our initial allocation to check for the results.