NAME	JACOB JOHN
REGISTER NO.	16BCE2205
E-MAIL	jacob.john2016@vitstudent.ac.in

LAB ASSESSMENT #6

SCENARIO – 1

Consider the following program, called mpi_sample2.c. Most parallel codes assign different tasks to different processors.

For example, parts of an input data set might be divided and processed by different processors, or a finite difference grid might be divided among the processors available. This means that the code needs to identify processors. In this example, processors are identified by rank - an integer from 0 to total number of processors - 1.

```
#include < mpi.h> /* PROVIDES THE BASIC MPI DEFINITION AND TYPES */ #include <
stdio.h>

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

int my_rank;
int size;
MPI_Init(&argc, &argv); /*START MPI */

/*DETERMINE RANK OF THIS PROCESSOR*/ MPI_Comm_rank(MPI_COMM_WORLD,
&my_rank);

/*DETERMINE TOTAL NUMBER OF PROCESSORS*/ MPI_Comm_size(MPI_COMM_WORLD,
&size);

printf("Hello world! I'm rank (processor number) %d of size %d\n", my_rank, size);

MPI_Finalize(); /* EXIT MPI */ }
```

1. Implement the above code in Microsoft Visual Studio
2. Build and Execute Its Logical Scenario
3. Depict the screenshots along with proper justification

Linux Execution

To compile this code, type:

```
mpicc -o simple1 mpi_sample1.c.c
```

Or

```
mpif77 -o simple1 mpi_sample1.c.f
```

To run this compiled code, type: mpirun -np 4 simple1

MS Studio Execution

1. Build the Code Using Menu Option

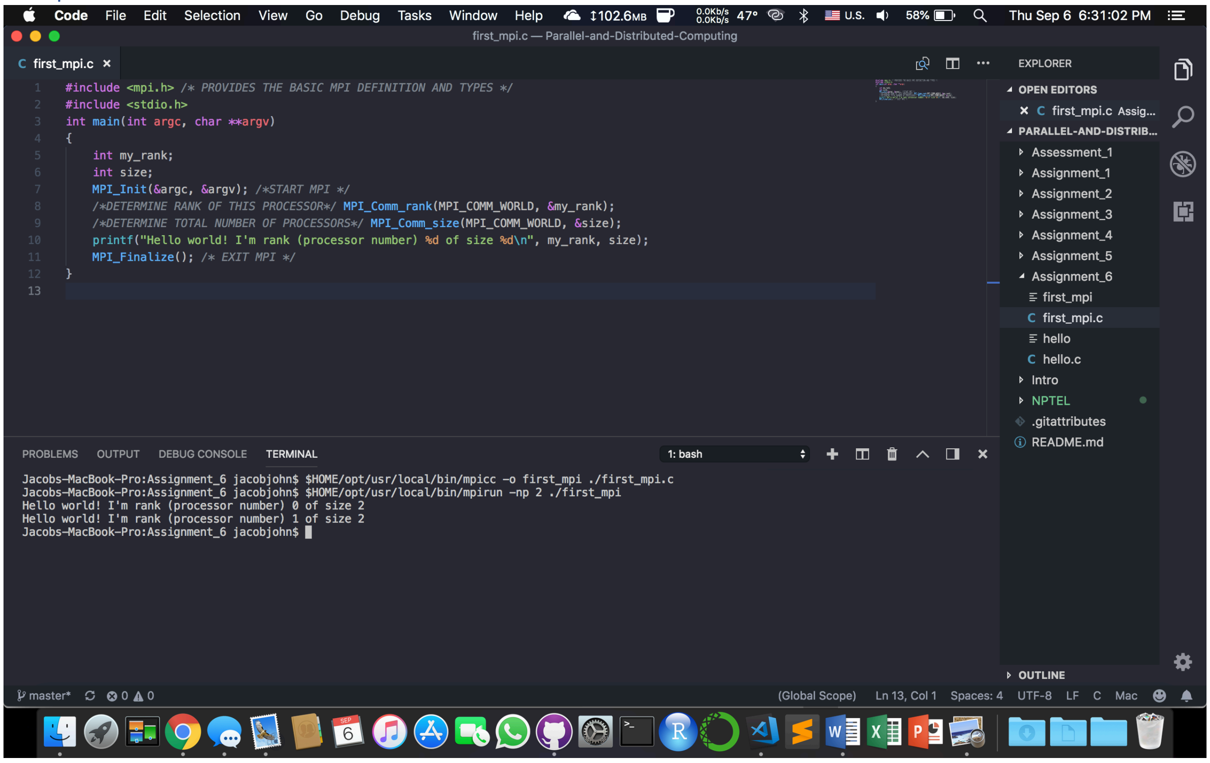
2. Run the program on the command line mpiexec -n 4 helloworld

[The syntax is almost the same as for the MPICH version of mpirun; instead of using -np to specify the number of processes, the switch -n is used.]

Code:

```c
#include < mpi.h> /* PROVIDES THE BASIC MPI DEFINITION AND TYPES */
#include < stdio.h>
int main(int argc, char **argv)
{
    int my_rank;
    int size;
    MPI_Init(&argc, &argv); /*START MPI */
    /*DETERMINE RANK OF THIS PROCESSOR*/ MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    /*DETERMINE TOTAL NUMBER OF PROCESSORS*/ MPI_Comm_size(MPI_COMM_WORLD, 
&size);
    printf("Hello world! I'm rank (processor number) %d of size %d\n", my_rank, 
size);
    MPI_Finalize(); /* EXIT MPI */
}
```

Output:



```c
#include <mpi.h> /* PROVIDES THE BASIC MPI DEFINITION AND TYPES */
#include <stdio.h>
int main(int argc, char **argv)
{
    int my_rank;
    int size;
    MPI_Init(&argc, &argv); /*START MPI */
    /*DETERMINE RANK OF THIS PROCESSOR*/ MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    /*DETERMINE TOTAL NUMBER OF PROCESSORS*/ MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello world! I'm rank (processor number) %d of size %d\n", my_rank, size);
    MPI_Finalize(); /* EXIT MPI */
}
```

```
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o first_mpi ./first_mpi.c
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./first_mpi
Hello world! I'm rank (processor number) 0 of size 2
Hello world! I'm rank (processor number) 1 of size 2
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ 
```

Execution

Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o first_mpi ./first_mpi.cJacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./first_mpiHello world! I'm rank (processor number) 0 of size 2
Hello world! I'm rank (processor number) 1 of size

SCENARIO – 2

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

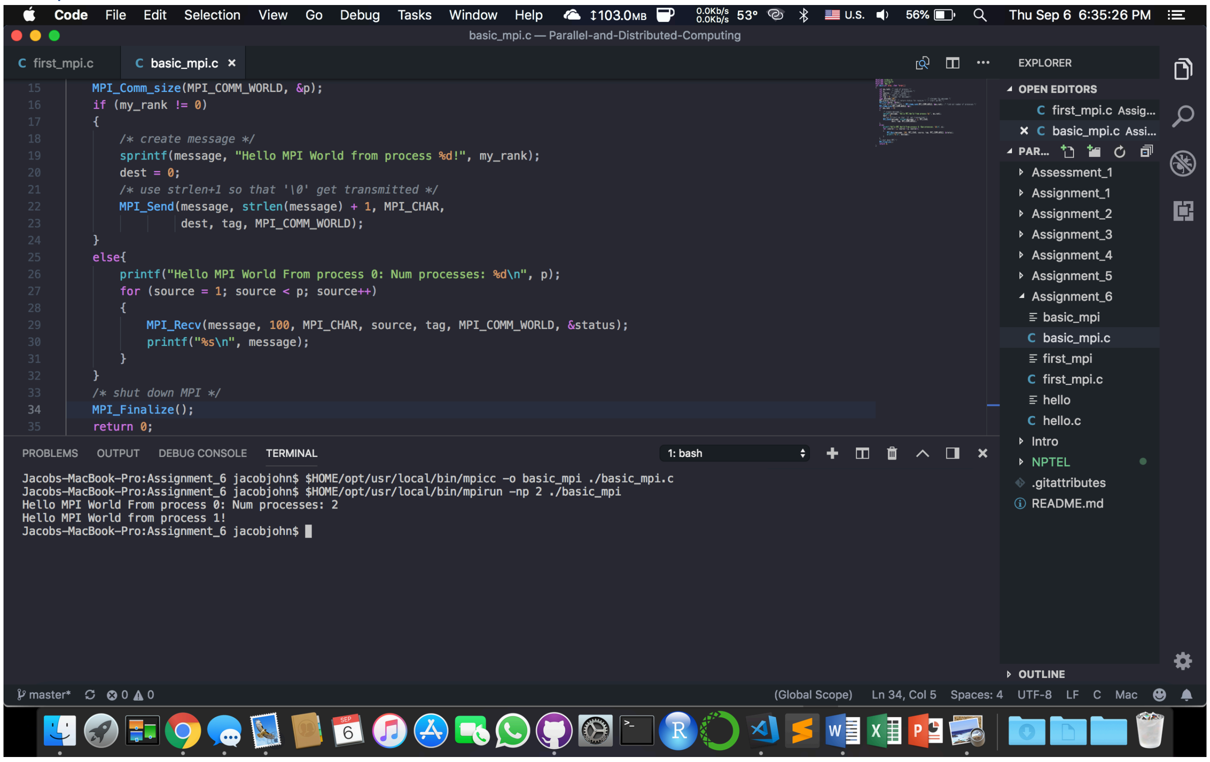
The new MPI calls are to MPI_Send and MPI_Recv and to MPI_Get_processor_name. The latter is a convenient way to get the name of the processor on which a process is running. MPI_Send and MPI_Recv can be understood by stepping back and considering the two requirements that must be satisfied to communicate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

Code:

```c
#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; /*tags 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, "Hello MPI World from process %d!", 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("Hello MPI World From process 0: Num processes: %d\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;
}
```

Output:



```
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    if (my_rank != 0)
    {
        /* create message */
        sprintf(message, "Hello MPI World from process %d!", 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("Hello MPI World From process 0: Num processes: %d\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;
```

```
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o basic_mpi ./basic_mpi.c
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./basic_mpi
Hello MPI World From process 0: Num processes: 2
Hello MPI World from process 1!
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$
```

Execution

```
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o basic_mpi
./basic_mpi.c
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./basic_mpi
Hello MPI World From process 0: Num processes: 2
Hello MPI World from process 1!
```

SCENARIO – 3

Write a simple MPI program to print a word 'Greetings' the processes. In order to explore its practical use, you are advised to read and understand the following statements.

1. MPI provides a set of send and receive functions that allow the communication of typed data with an associated tag.
2. Typing of the message contents is necessary for heterogeneous support - the type information is needed so a correct data representation conversion can be performed as data is sent from one architecture to anothers.
3. The tag allows selectivity of messages at the receiving endone can receive on a particular tag or one can wildcard this quantity allowing reception of messages with any tag.
4. Message selectivity on the source process of the message is also provided.

Brief about your approach

For the example of process 0 sending a message to process 1. The code executes on both process 0 and process 1. Process 0 sends a character string using MPI_Send(). The first three parameters of the send call specify the data to be sent: 1. The outgoing data is to be taken from msg; 2. It consists of strlen (msg) + 1 entries each of type MPI CHAR (The string "Hello There" contains strlen (msg) =11 significant characters. In addition, we are also sending the '\0' string terminator character. The fourth parameter specifies the message destination which is process 1. The fifth parameter specifies the message tag. Finally the last parameter is a communicator that specifies a communication domain for this communication.

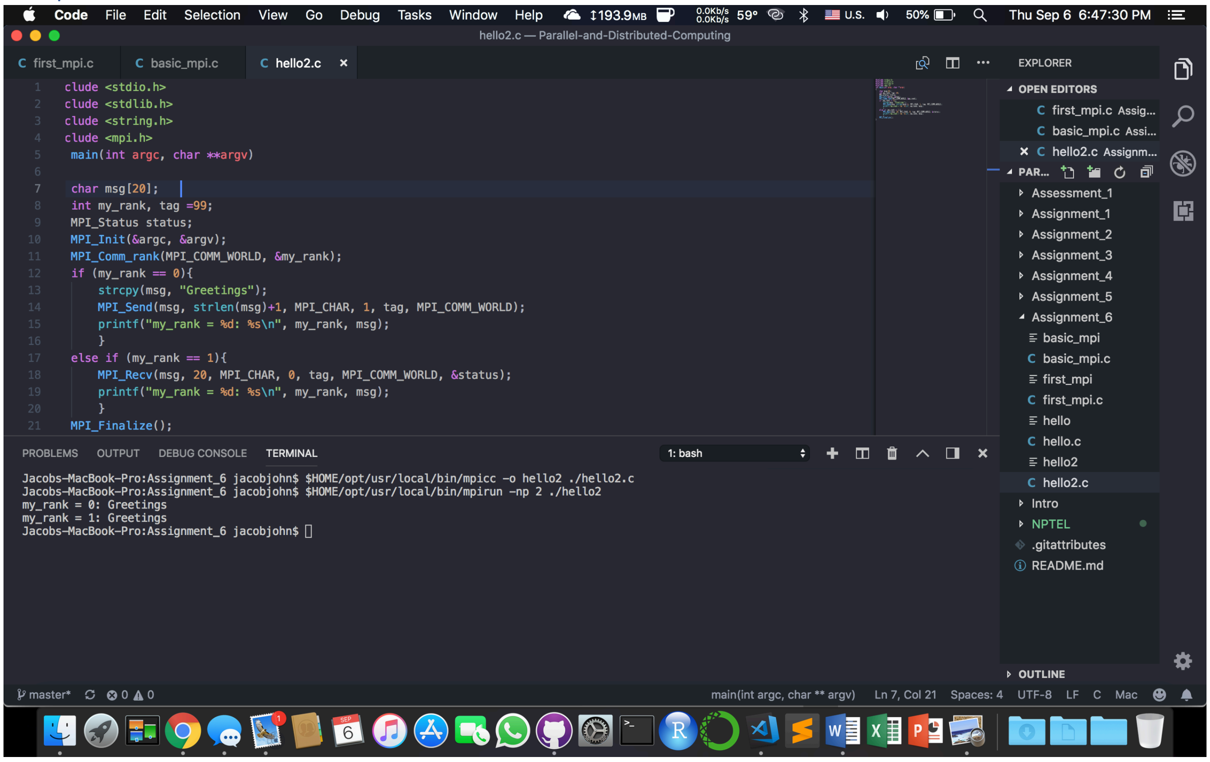
Among other things a communicator serves to define a set of processes that can be contacted. Each such process is labelled by a process rank. Process ranks are integers and are discovered by inquiry to a communicator (MPI_Comm_rank ()). MPI_COMM_WORLD is a default communicator provided upon start-up that defines an initial communication domain for all the processes that participate in the computation.

The receiving process specified that the incoming data was to be placed in msg and that it has a maximum size of 20 entries of type MPI_CHAR. The variable status set by MPI_Recv() gives information on the source and tag of the message and how many elements were actually received. For example, the receiver can examine this variable to find out the actual length of the character string received.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>
int main(int argc, char **argv)
{
    char msg[20];
    int my_rank, tag =99;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    if (my_rank == 0){
        strcpy(msg, "Greetings");
        MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
        printf("my_rank = %d: %s\n", my_rank, msg);
        }
    else if (my_rank == 1){
        MPI_Recv(msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
        printf("my_rank = %d: %s\n", my_rank, msg);
        }
    MPI_Finalize();
}
```

Output:



```
clude <stdio.h>
clude <stdlib.h>
clude <string.h>
clude <mpi.h>
 main(int argc, char **argv)

 char msg[20];
 int my_rank, tag =99;
 MPI_Status status;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 if (my_rank == 0){
     strcpy(msg, "Greetings");
     MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
     printf("my_rank = %d: %s\n", my_rank, msg);
     }
 else if (my_rank == 1){
     MPI_Recv(msg, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
     printf("my_rank = %d: %s\n", my_rank, msg);
     }
 MPI_Finalize();
```

```
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o hello2 ./hello2.c
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./hello2
my_rank = 0: Greetings
my_rank = 1: Greetings
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$
```

Execution

```
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpicc -o hello2 ./hello2.c
Jacobs-MacBook-Pro:Assignment_6 jacobjohn$ $HOME/opt/usr/local/bin/mpirun -np 2 ./hello2
my_rank = 0: Greetings
my_rank = 1: Greetings
```