**High Performance Computing Lab**

**Practical No. 7**

**2020BTECS00049, B3**

**Siddhi Hagawane**

**Title of practical:**

Installation of MPI & Implementation of basic functions of MPI

installtion of mpi on ubantu.

Get done by this command on terminal:- sudo apt install mpich

mpi compiler:- mpicc is used to compile a program in c.

**Problem Statement 1:**

Implement a simple hello world program by setting number of processes equal to 10

**Screenshots:**

MPI code for hello program in c

// hello\_mpi.c

#include <stdio.h>

#include <mpi.h>

int main(int argc, char \*argv[]) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

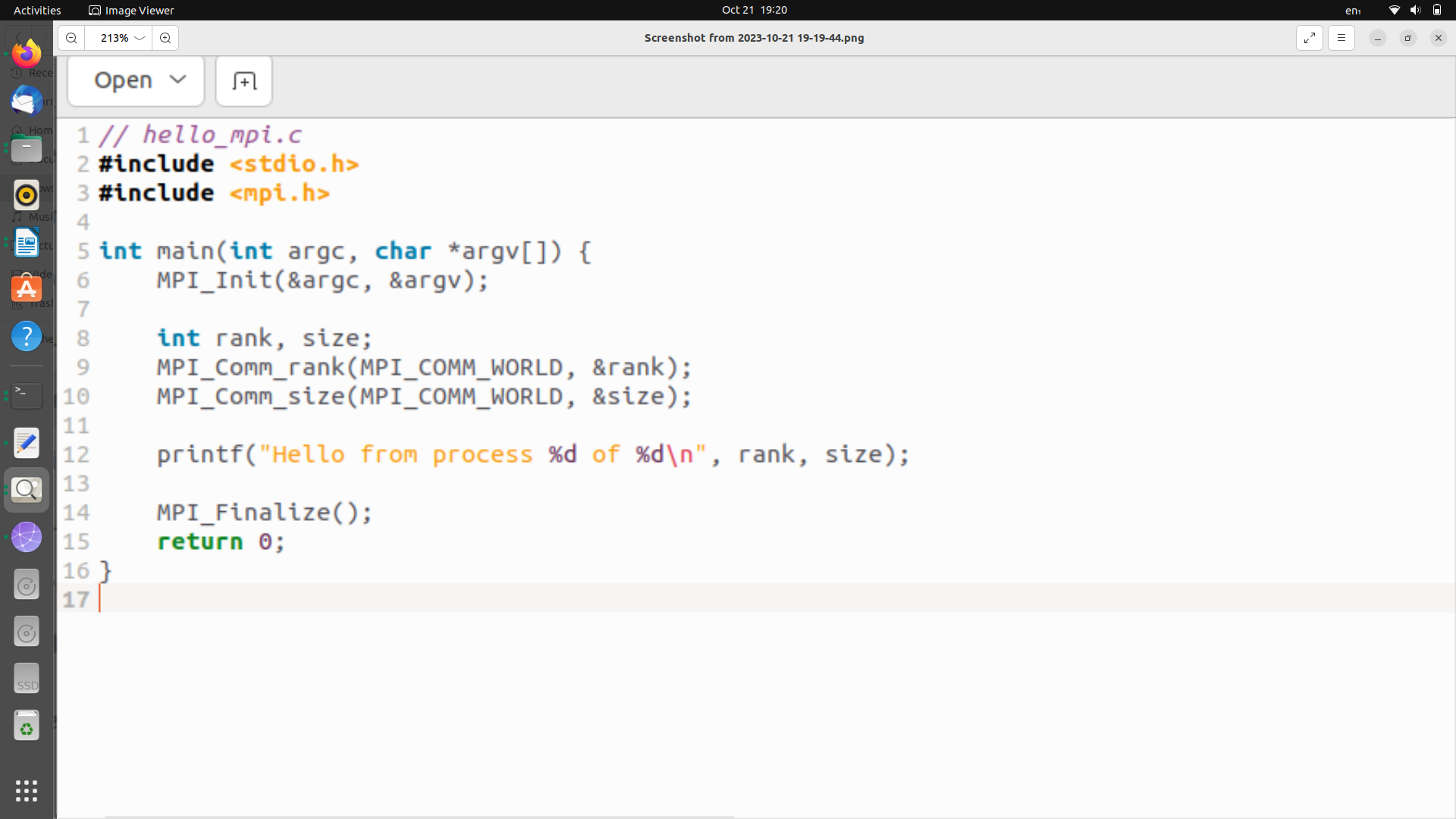
printf("Hello from process %d of %d\n", rank, size);

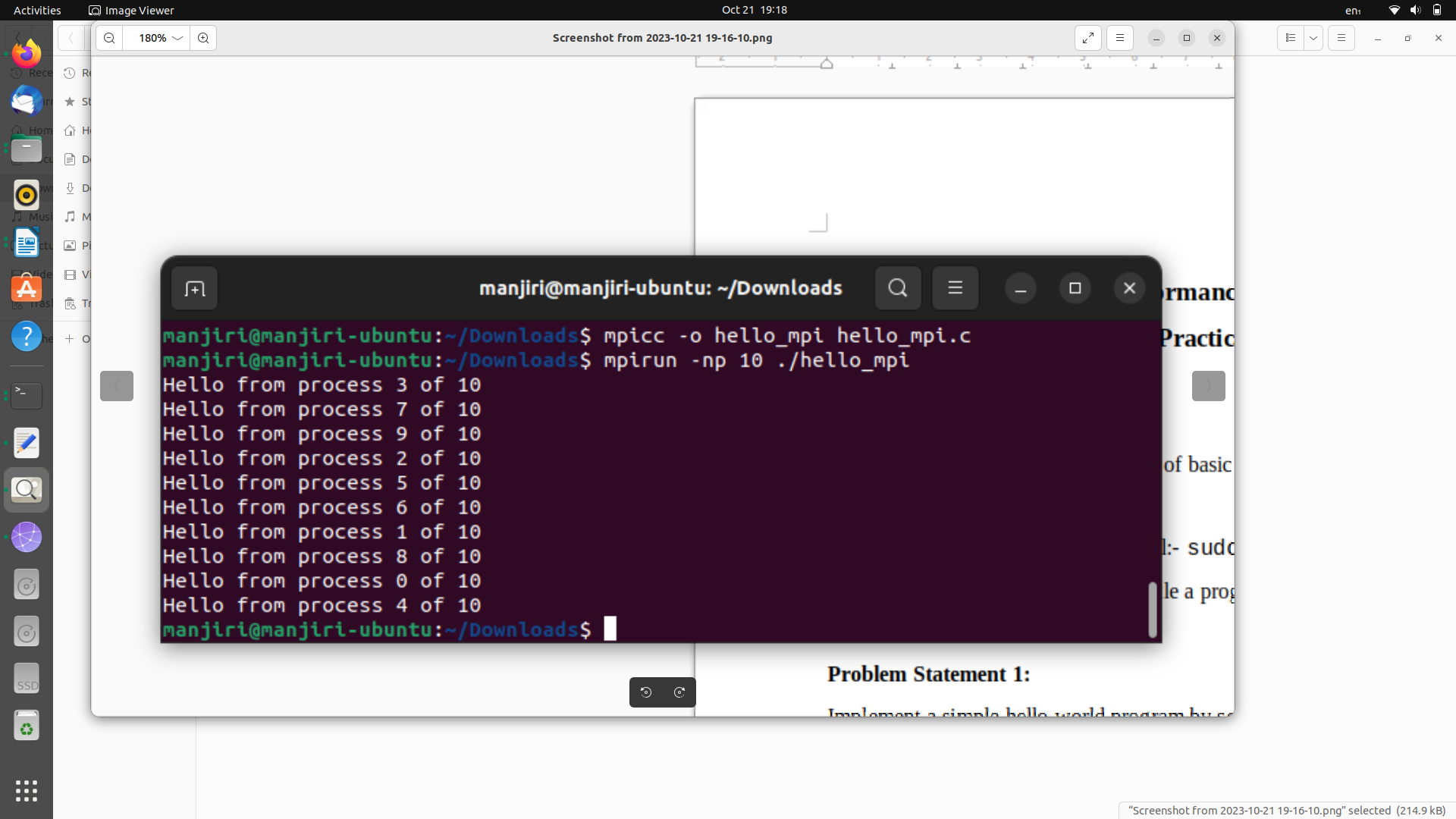
MPI\_Finalize();

return 0;

**}**

**Code in c**

**Output:**

****

**Explaination:**

#include <mpi.h>: This line includes the MPI header file, which provides the necessary functions and data types for writing MPI programs.

int main(int argc, char \*argv[]) {

* int main(int argc, char \*argv[]): This is the main function of the program, which is the entry point for the application. It takes two arguments: argc is the number of command-line arguments, and argv is an array of strings that contains the command-line arguments.

MPI\_Init(&argc, &argv);

* MPI\_Init(&argc, &argv): This initializes the MPI environment. It's necessary to call this function at the beginning of an MPI program to set up the necessary data structures and resources.

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

* int rank, size;: These variables will store the rank and size of each process.
* MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);: This function retrieves the rank (unique identifier) of the current process within the MPI communicator MPI\_COMM\_WORLD.
* MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);: This function retrieves the size (total number of processes) within the MPI communicator MPI\_COMM\_WORLD.

printf("Hello from process %d of %d\n", rank, size);

* printf("Hello from process %d of %d\n", rank, size);: This line prints a message to the console. It includes the rank and size of the current process, so when you run the program with multiple processes, each process will identify itself with a unique rank and display its rank and the total number of processes.

MPI\_Finalize();

* MPI\_Finalize();: This function finalizes the MPI environment, releasing resources and cleaning up after the program has finished running.

**Information 1:**

The primary purpose of MPI in this program is to demonstrate parallelism, where multiple processes can execute the same code but with different data. Each process can communicate and synchronize with others, which is essential for more complex parallel and distributed applications.

In this simple example, MPI is used to ->

1. Initializing the MPI environment using MPI\_Init.
2. Retrieving the rank and size of the MPI process.
3. Printing a message that includes the rank and size to identify each process.
4. Finalizing the MPI environment using MPI\_Finalize.

**Problem Statement 2:**

Implement a program to display rank and communicator group of five processes

**Screenshots:**

**code in c:**

#include <mpi.h>

#include <stdio.h>

int main(int argc, char \*argv[]) {

MPI\_Init(&argc, &argv);

int rank;

MPI\_Group group;

MPI\_Comm\_group(MPI\_COMM\_WORLD, &group);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

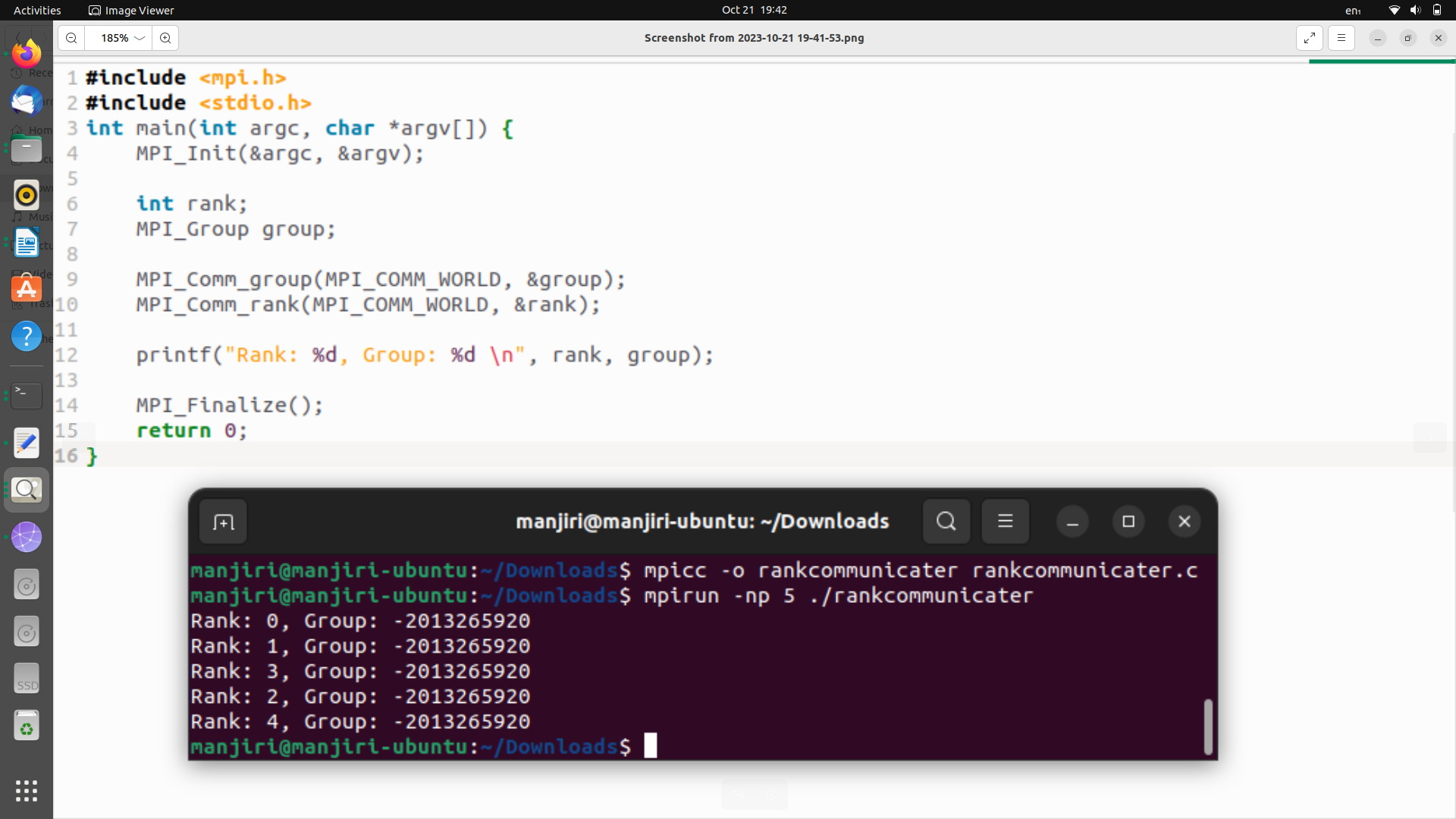
printf("Rank: %d, Group: %d \n", rank, group);

MPI\_Finalize();

return 0;

}

**OutPut:**

****

**Information:**

The code begins with including the necessary header files for MPI and standard I/O. It initializes the MPI environment using MPI\_Init with command-line arguments argc and argv.

The MPI\_Init function initializes the MPI environment and is required at the start of every MPI program.

Two variables are declared: rank to store the rank of the process and group to store the group information of the process.

MPI\_Comm\_group retrieves the group information of the communicator MPI\_COMM\_WORLD and stores it in the group variable.

MPI\_COMM\_WORLD is a predefined communicator that includes all the processes in the MPI job. It represents the default communication group that contains all processes.

MPI\_Comm\_rank retrieves the rank of the current process within the MPI\_COMM\_WORLD communicator and stores it in the rank variable.

The program then prints the rank and group information. However, there is an issue in this line. The group variable should not be printed directly as an integer because it represents an MPI group, which is not a single integer value. To display the groupinformation, it would print some memory address that represents the MPI\_Group structure.

**GitHub Link:**

<https://github.com/Siddhish16/HPC-Assignments>