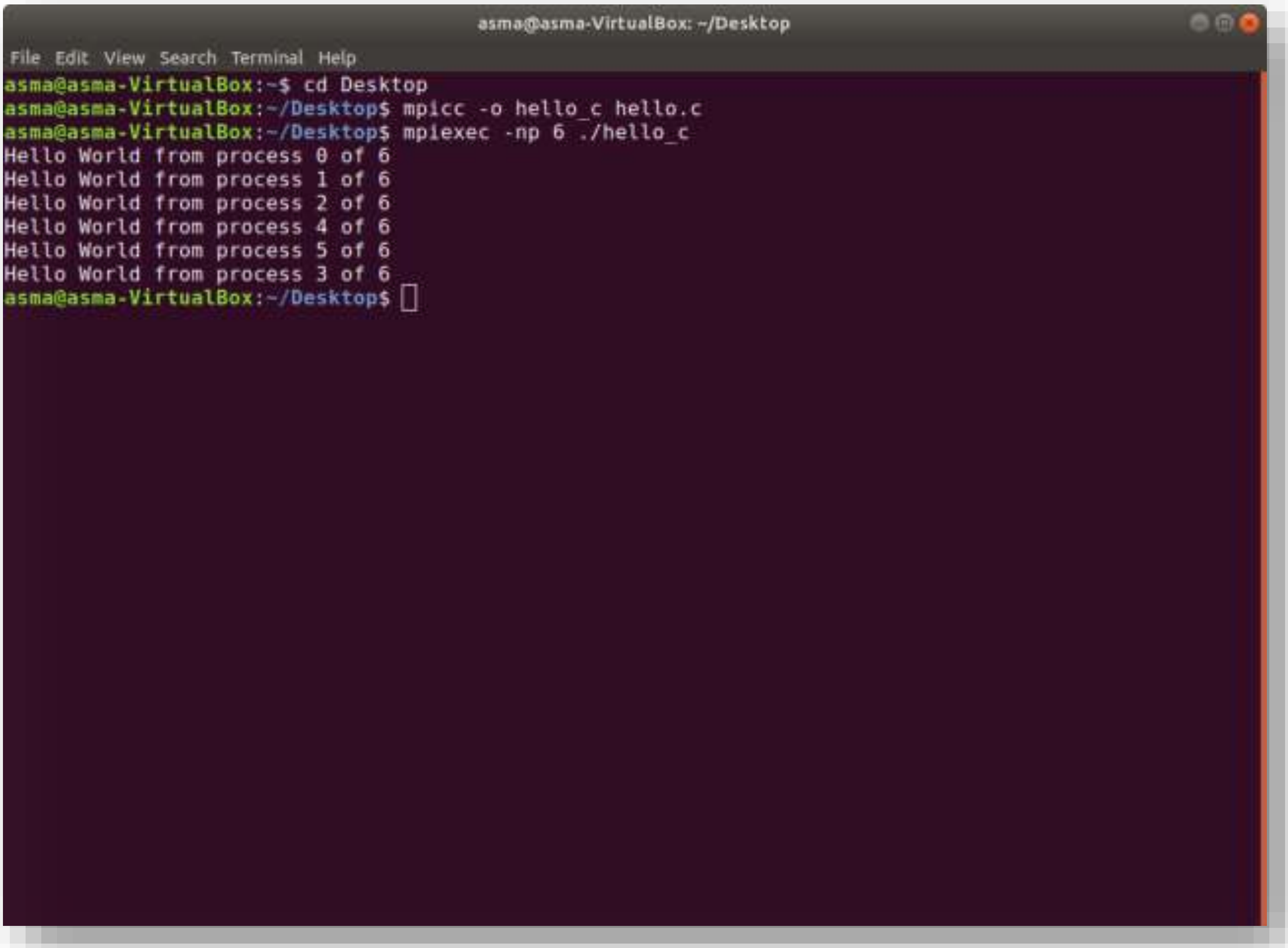
Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 1

Aim: Execution of Simple Hello Word program on MPI platform.

```
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] )
{
    int rank, size;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    printf( "Hello World from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```



asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

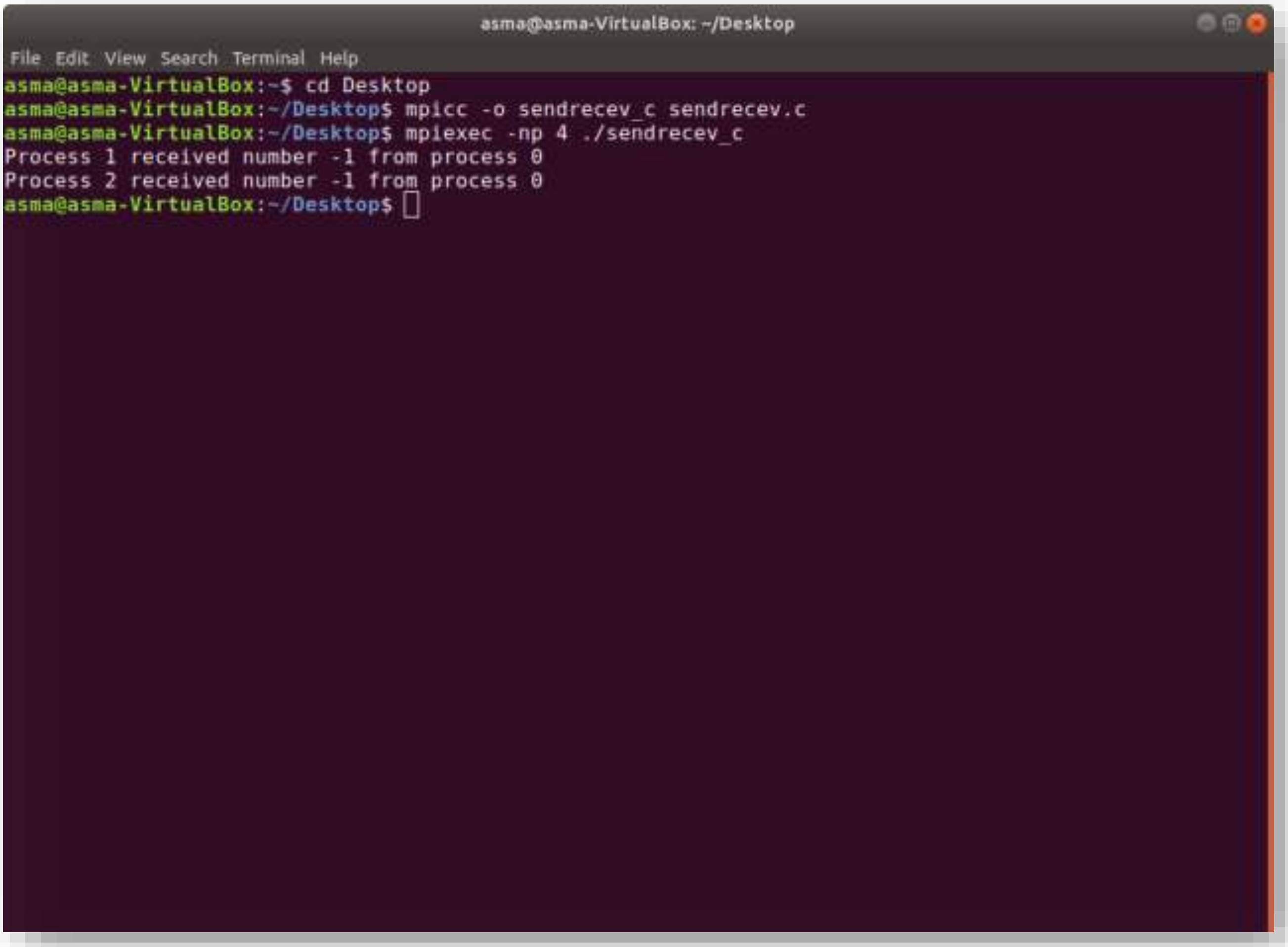
```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o hello_c hello.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 6 ./hello_c
Hello World from process 0 of 6
Hello World from process 1 of 6
Hello World from process 2 of 6
Hello World from process 4 of 6
Hello World from process 5 of 6
Hello World from process 3 of 6
asma@asma-VirtualBox:~/Desktop$
```

Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 2

Aim: a. Program to send and receive data to/from processors using MPI.
b. Program illustrating Broadcast of data using MPI.

```
#include "mpi.h"
#include<stdio.h>


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

MPI_Init(NULL, NULL);
// Find out rank, size
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int number;
if (world_rank == 0) {
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
MPI_Send(&number, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);
} else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n",
           number);
}
 if (world_rank == 2) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("Process 2 received number %d from process 0\n",
           number);
}
MPI_Finalize();
return 0;
}
```

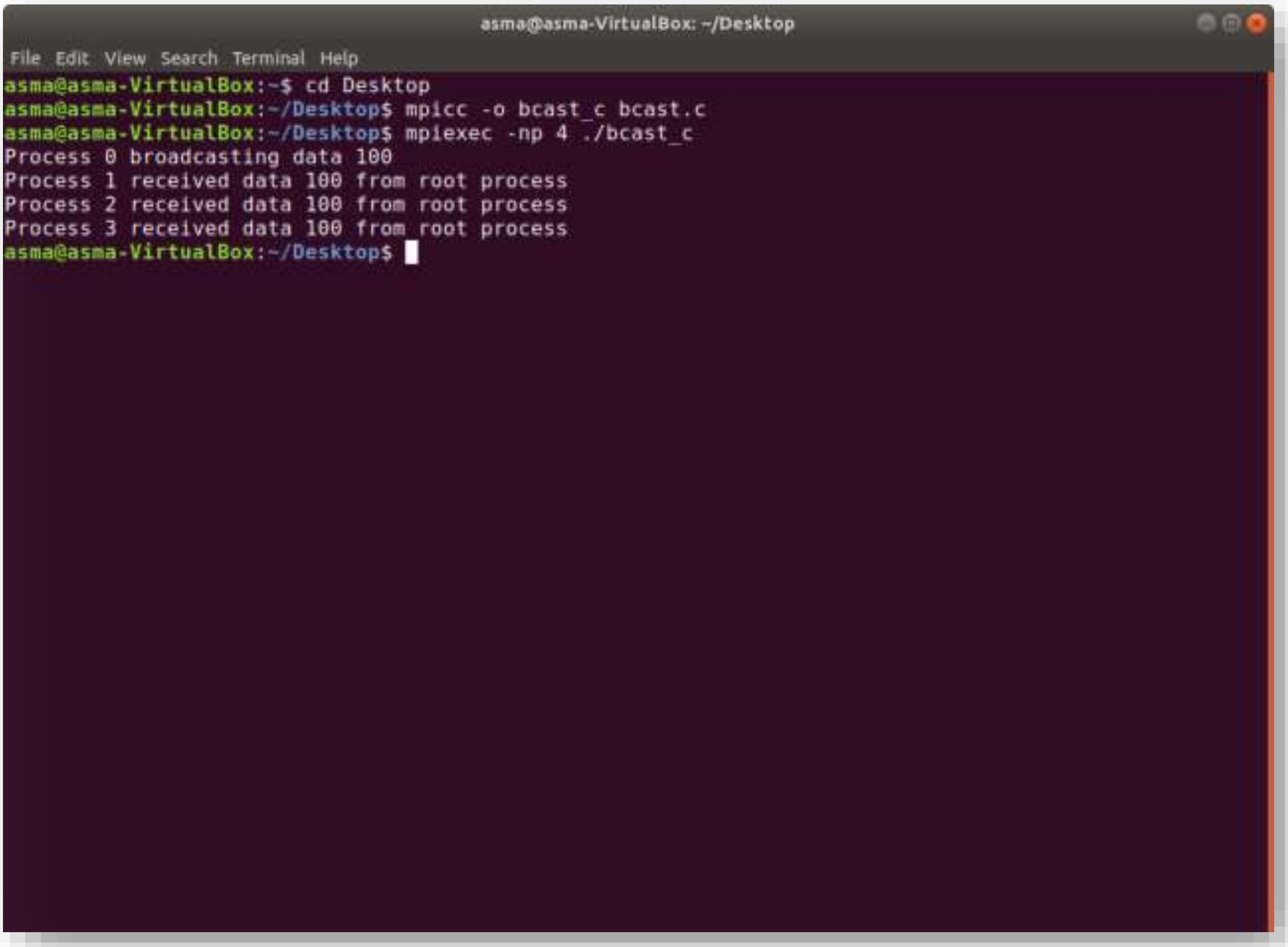


asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o sendrecev_c sendrecev.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 4 ./sendrecev_c
Process 1 received number -1 from process 0
Process 2 received number -1 from process 0
asma@asma-VirtualBox:~/Desktop$ 
```

b. Program illustrating Broadcast of data using MPI.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

void my_bcast(void* data, int count, MPI_Datatype datatype, int
root,
              MPI_Comm communicator) {
  int world_rank;
  MPI_Comm_rank(communicator, &world_rank);
  int world_size;
  MPI_Comm_size(communicator, &world_size);

  if (world_rank == root) {
    // If we are the root process, send our data to everyone
    int i;
    for (i = 0; i < world_size; i++) {
      if (i != world_rank) {
        MPI_Send(data, count, datatype, i, 0, communicator);
      }
    }
  } else {
    // If we are a receiver process, receive the data from the
root
    MPI_Recv(data, count, datatype, root, 0, communicator,
MPI_STATUS_IGNORE);
  }
}

int main(int argc, char** argv) {
  MPI_Init(NULL, NULL);

  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  int data;
  if (world_rank == 0) {
    data = 100;
    printf("Process 0 broadcasting data %d\n", data);
    my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
  } else {
    my_bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("Process %d received data %d from root process\n",
world_rank, data);
  }

  MPI_Finalize();
}
```


asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

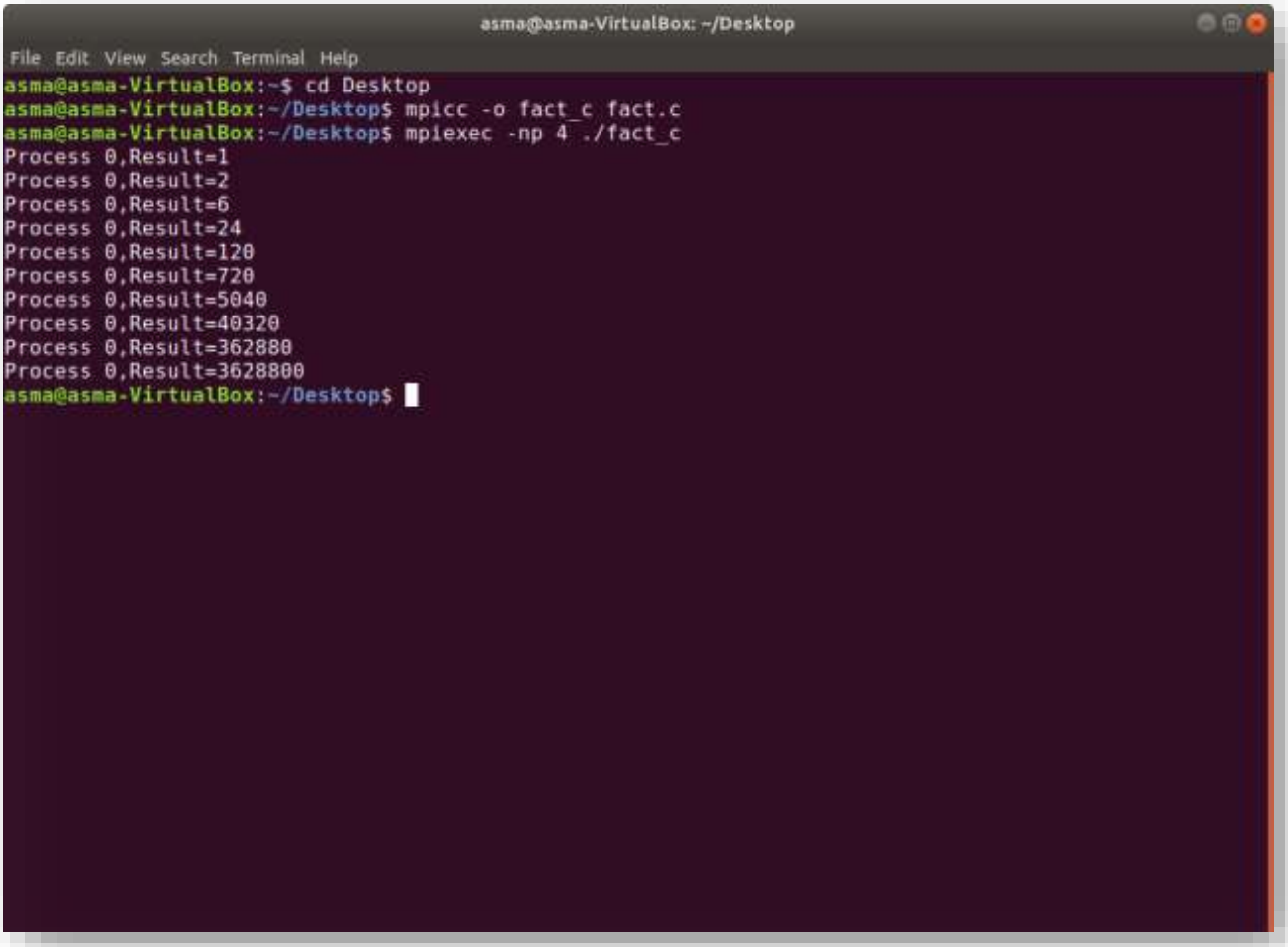
```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o bcast_c bcast.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 4 ./bcast_c
Process 0 broadcasting data 100
Process 1 received data 100 from root process
Process 2 received data 100 from root process
Process 3 received data 100 from root process
asma@asma-VirtualBox:~/Desktop$ 
```

Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 3

Aim: To calculate factorial of a number.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mpi.h>
int main (int argc, char ** argv){
    int rank, size,tag=100;
    MPI_Init (&argc, &argv);  /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);    /* get current 
process id */

    if (rank == 0)
    {
        int a[10]={1,2,3,4,5,6,7,8,9,10}; // the array with the 
values to calculate the factorial
        int fact[10] = {0}; // the array to store the results
        MPI_Send(a, 10, MPI_INT,1,tag, MPI_COMM_WORLD); // the 
values
        MPI_Recv(fact, 10, MPI_INT,1,tag, 
MPI_COMM_WORLD,MPI_STATUSES_IGNORE); // wait for the result
        for(int i = 0; i < 10; i++) // print the results;
           printf("Process %d,Result=%d\n",rank, fact[i]);
    }
    else if (rank == 1)
    {
       int a[10] = {0};
       int fact[10] = {0};
       MPI_Recv(a, 10, MPI_INT,0,tag, 
MPI_COMM_WORLD,MPI_STATUSES_IGNORE);
       for(int i = 0; i < 10; i++){
           int f = 1;
           for (int k = 1; k <= a[i]; ++k) // Calculate the 
factorials
                f *= k;
           fact[i] = f;
       }
       MPI_Send(fact,10, MPI_INT,0,tag, MPI_COMM_WORLD); // send 
the factorials to process 0
    }
    MPI_Comm_size (MPI_COMM_WORLD, &size);    /* get number of 
processes */
    MPI_Finalize();
    return 0;
}
```



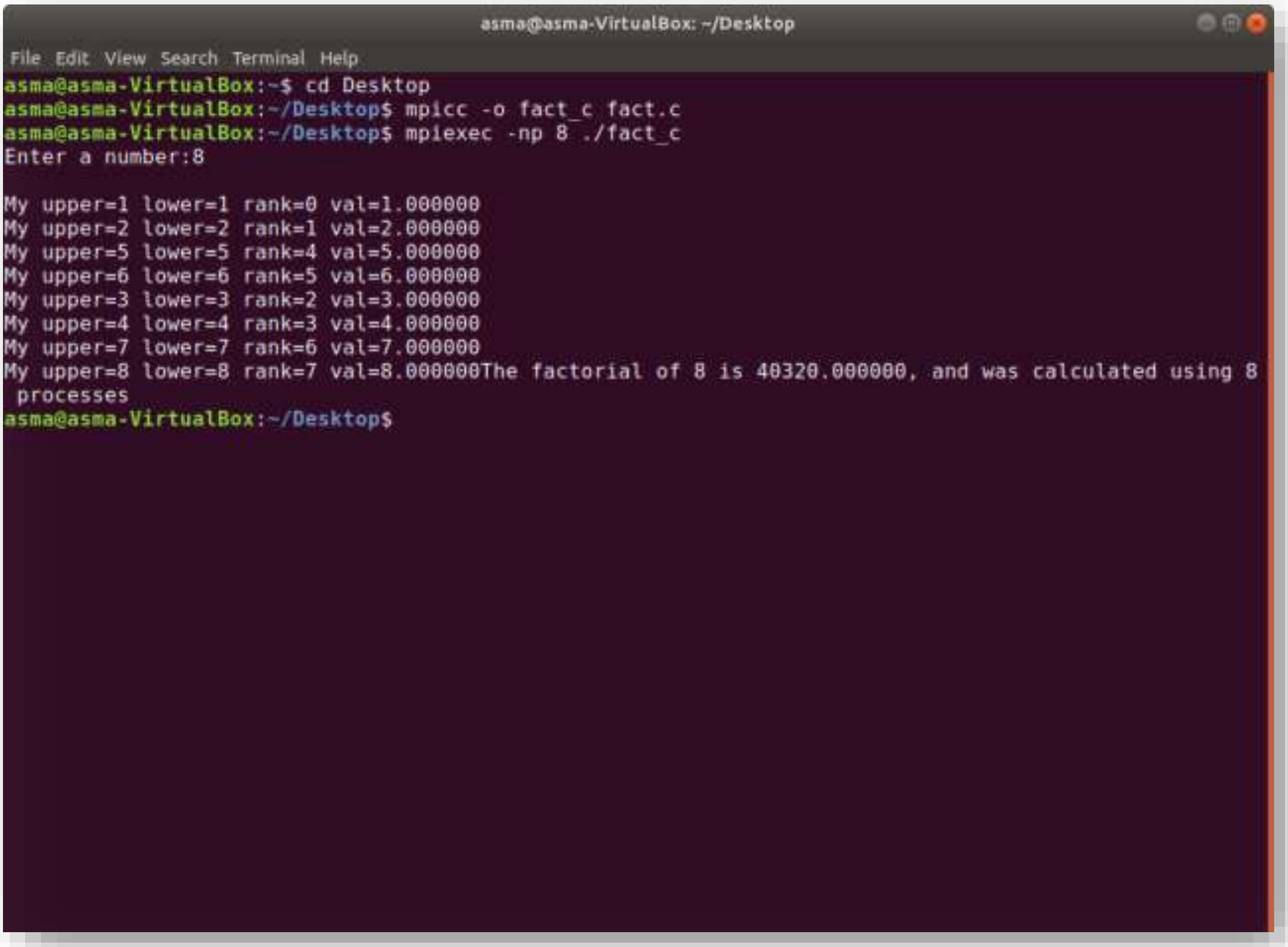
asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o fact_c fact.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 4 ./fact_c
Process 0,Result=1
Process 0,Result=2
Process 0,Result=6
Process 0,Result=24
Process 0,Result=120
Process 0,Result=720
Process 0,Result=5040
Process 0,Result=40320
Process 0,Result=362880
Process 0,Result=3628800
asma@asma-VirtualBox:~/Desktop$ 
```

/* simple parallel factorial calculator. Only useful
 * to illustrate collective communication :)
 */

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[]){
 int myRank;
 int size;
 int fact;
 int lower,upper;
 int i;
 double local_result = 1.0;
 double total;

 /* initialize MPI */
 MPI_Init(&argc,&argv);
 /* get my rank and the size of the communicator */
 MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
      /* get the input. (only if i have rank 0) */
      if(myRank==0){
      printf("Enter a number:");
      scanf("%d",&fact);
      }
      /* since only the process with rank 0 has the input,
      * we must pass it to all the other processes. */

      MPI_Bcast(&fact, /* in/out parameter */
      1, /* count */
      MPI_INT, /* datatype */
      0, /* root */
      MPI_COMM_WORLD); /* communicator */

      /* calculate the upper and lower boundaries
      * for each process
      */
      if(myRank==0){
      lower = 1;
      }else
      lower = myRank * (fact / size) + 1;
      if(myRank==(size-1))
      upper = fact;
      else
      upper = (myRank + 1) * (fact / size);

      /* now that we know upper and lower, do the
      * multiplication in our local area
      */
      for(i=lower;i<=upper;i++){
      local_result = local_result * (double)i;
printf("\nMy upper=%d lower=%d rank=%d
val=%lf",upper,lower,myRank,local_result);
 }

      /* combine all the local results by multiplying them
      * together
      */
      MPI_Reduce(&local_result, /* operand */
      &total, /* result */
      1, /* count */
      MPI_DOUBLE, /* datatype */
      MPI_PROD, /* operator */
      0, /* root rank */
      MPI_COMM_WORLD); /* communicator */

      /* give the output to the user */
      if(myRank==0){
      printf("The factorial of %d is %lf, and was calculated
using %d processes\n",fact,total,size);
      }

      /* shut down MPI */
      MPI_Finalize();

      return 0;
     }
```



asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

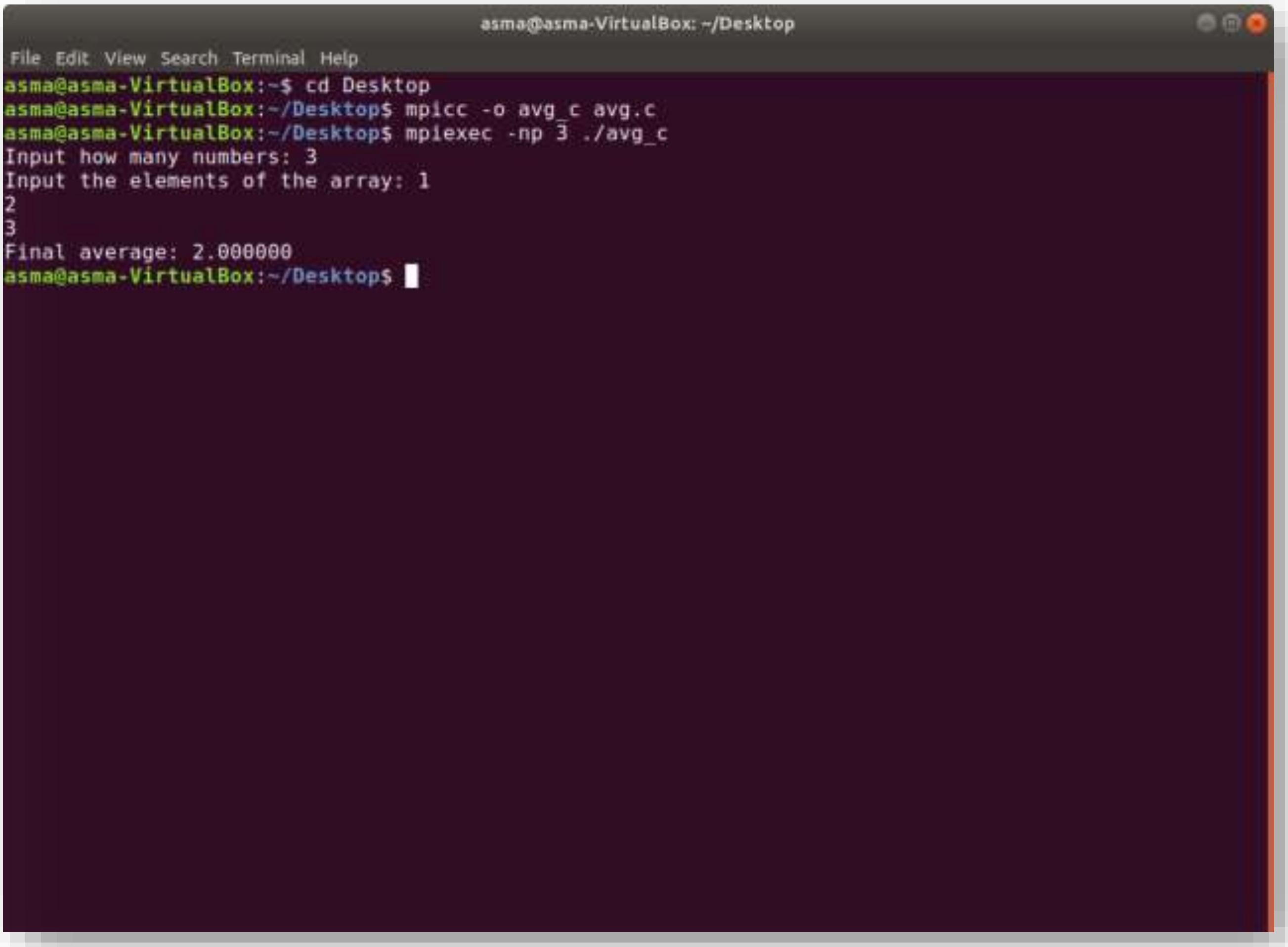
```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o fact_c fact.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 8 ./fact_c
Enter a number:8

My upper=1 lower=1 rank=0 val=1.000000
My upper=2 lower=2 rank=1 val=2.000000
My upper=5 lower=5 rank=4 val=5.000000
My upper=6 lower=6 rank=5 val=6.000000
My upper=3 lower=3 rank=2 val=3.000000
My upper=4 lower=4 rank=3 val=4.000000
My upper=7 lower=7 rank=6 val=7.000000
My upper=8 lower=8 rank=7 val=8.000000The factorial of 8 is 40320.000000, and was calculated using 8
 processes
asma@asma-VirtualBox:~/Desktop$
```

Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 4

Aim: a. To implement Average of an array.
b. To implement ring algorithm.

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char** argv){
    int my_rank;
    int total_processes;
    int root = 0;
    int data[100];
    int data_loc[100];
    float final_res[100];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &total_processes);

    int input_size = 0;
    if (my_rank == 0){
       printf("Input how many numbers: ");
       scanf("%d", &input_size);

       printf("Input the elements of the array: ");
       for(int i=0; i<input_size; i++){
           scanf("%d", &data[i]);
       }
    }

    MPI_Bcast(&input_size, 1, MPI_INT, root, MPI_COMM_WORLD);

    int loc_num = input_size/total_processes;

    MPI_Scatter(&data, loc_num, MPI_INT, data_loc, loc_num,
MPI_INT, root, MPI_COMM_WORLD);

    int loc_sum = 0;
    for(int i=0; i< loc_num; i++)
        loc_sum += data_loc[i];
    float loc_avg = (float) loc_sum / (float) loc_num;
    MPI_Gather(&loc_avg, 1, MPI_FLOAT, final_res, 1, MPI_FLOAT,
root, MPI_COMM_WORLD);

    if(my_rank==0){
      float fin = 0;
      for(int i=0; i<total_processes; i++)
         fin += final_res[i];
      float avg = fin / (float) total_processes;
      printf("Final average: %f \n", avg);
    }
    MPI_Finalize();
    return 0;
}
```

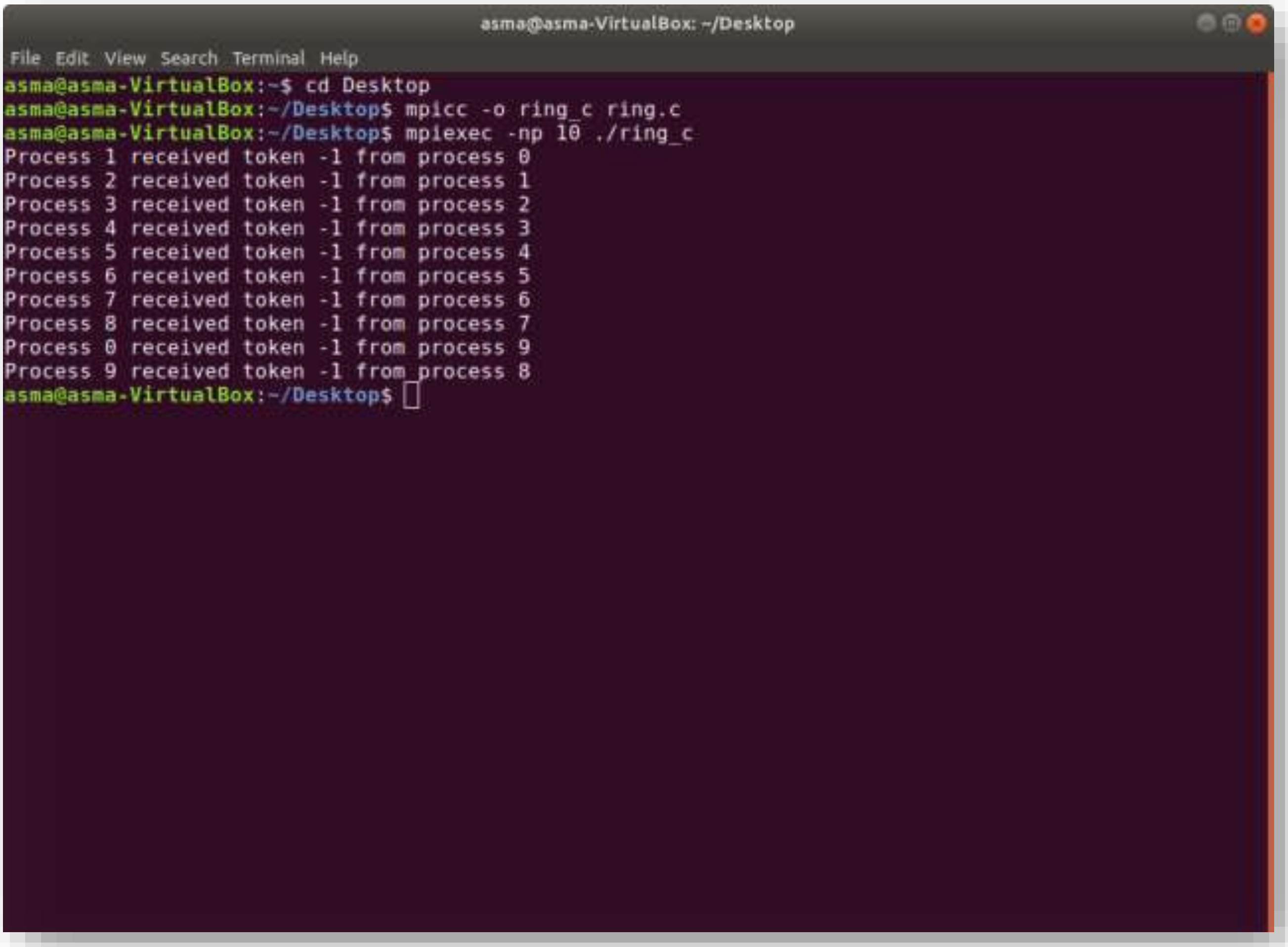


asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o avg_c avg.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 3 ./avg_c
Input how many numbers: 3
Input the elements of the array: 1
2
3
Final average: 2.000000
asma@asma-VirtualBox:~/Desktop$ 
```

b. To implement ring algorithm.

```
#include "mpi.h"
#include<stdio.h>


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

MPI_Init(&argc,&argv);
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int token;
if (world_rank != 0) {
    MPI_Recv(&token, 1, MPI_INT, world_rank - 1, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n",
           world_rank, token, world_rank - 1);
} else {
    // Set the token's value if you are process 0
    token = -1;
}
MPI_Send(&token, 1, MPI_INT, (world_rank + 1) % world_size,
         0, MPI_COMM_WORLD);

// Now process 0 can receive from the last process.
if (world_rank == 0) {
    MPI_Recv(&token, 1, MPI_INT, world_size - 1, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d received token %d from process %d\n",
           world_rank, token, world_size - 1);
}
MPI_Finalize();
return 0;
}
```



asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

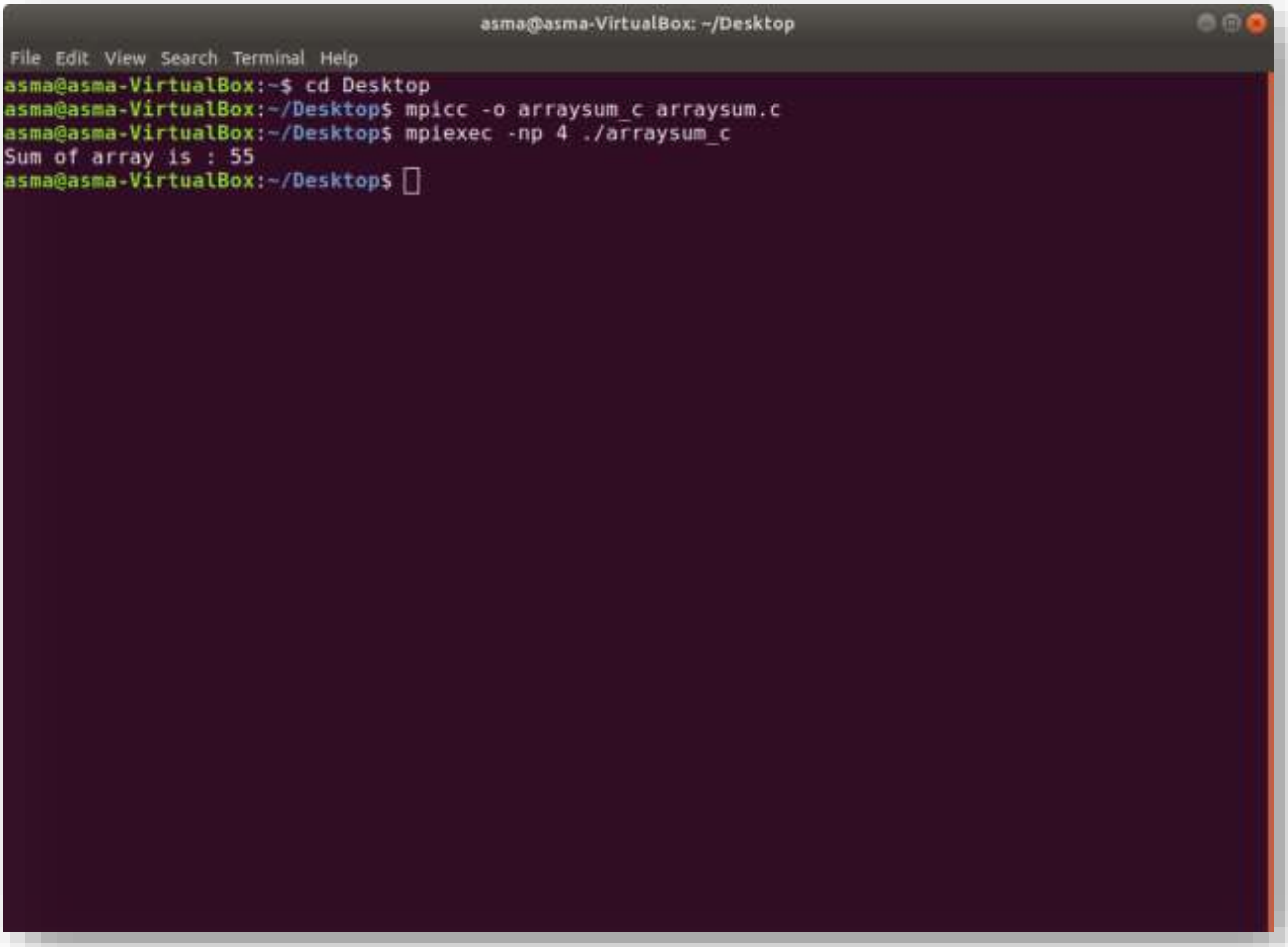
```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o ring_c ring.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 10 ./ring_c
Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 4 received token -1 from process 3
Process 5 received token -1 from process 4
Process 6 received token -1 from process 5
Process 7 received token -1 from process 6
Process 8 received token -1 from process 7
Process 0 received token -1 from process 9
Process 9 received token -1 from process 8
asma@asma-VirtualBox:~/Desktop$ 
```

Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 5

Aim: To find sum of a one-dimensional Array.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// size of array
#define n 10

int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Temporary array for slave process
int a2[1000];

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

    int pid, np,
        elements_per_process,
        n_elements_recieved;
    // np -> no. of processes
    // pid -> process id

    MPI_Status status;

    // Creation of parallel processes
    MPI_Init(&argc, &argv);

    // find out process ID,
    // and how many processes were started
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    // master process
    if (pid == 0) {
        int index, i;
        elements_per_process = n / np;

        // check if more than 1 processes are run
        if (np > 1) {
            // distributes the portion of array
            // to child processes to calculate
            // their partial sums
            for (i = 1; i < np - 1; i++) {
                index = i * elements_per_process;

                MPI_Send(&elements_per_process,
                         1, MPI_INT, i, 0,
                         MPI_COMM_WORLD);
                MPI_Send(&a[index],
                         elements_per_process,
                         MPI_INT, i, 0,
                         MPI_COMM_WORLD);
            }

            // last process adds remaining elements
            index = i * elements_per_process;
            int elements_left = n - index;

            MPI_Send(&elements_left,
                     1, MPI_INT,
                     i, 0,
                     MPI_COMM_WORLD);
            MPI_Send(&a[index],
                     elements_left,
                     MPI_INT, i, 0,
                     MPI_COMM_WORLD);
        }

        // master process add its own sub array
        int sum = 0;
        for (i = 0; i < elements_per_process; i++)
            sum += a[i];

        // collects partial sums from other processes
        int tmp;
        for (i = 1; i < np; i++) {
            MPI_Recv(&tmp, 1, MPI_INT,
                     MPI_ANY_SOURCE, 0,
                     MPI_COMM_WORLD,
                     &status);
            int sender = status.MPI_SOURCE;

            sum += tmp;
        }

        // prints the final sum of array
        printf("Sum of array is : %d\n", sum);
    }
    // slave processes
    else {
        MPI_Recv(&n_elements_recieved,
                 1, MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // stores the received array segment
        // in local array a2
        MPI_Recv(&a2, n_elements_recieved,
                 MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        // calculates its partial sum
        int partial_sum = 0;
        for (int i = 0; i < n_elements_recieved; i++)
            partial_sum += a2[i];

        // sends the partial sum to the root process
        MPI_Send(&partial_sum, 1, MPI_INT,
                 0, 0, MPI_COMM_WORLD);
    }

    // cleans up all MPI state before exit of process
    MPI_Finalize();

    return 0;
}
```



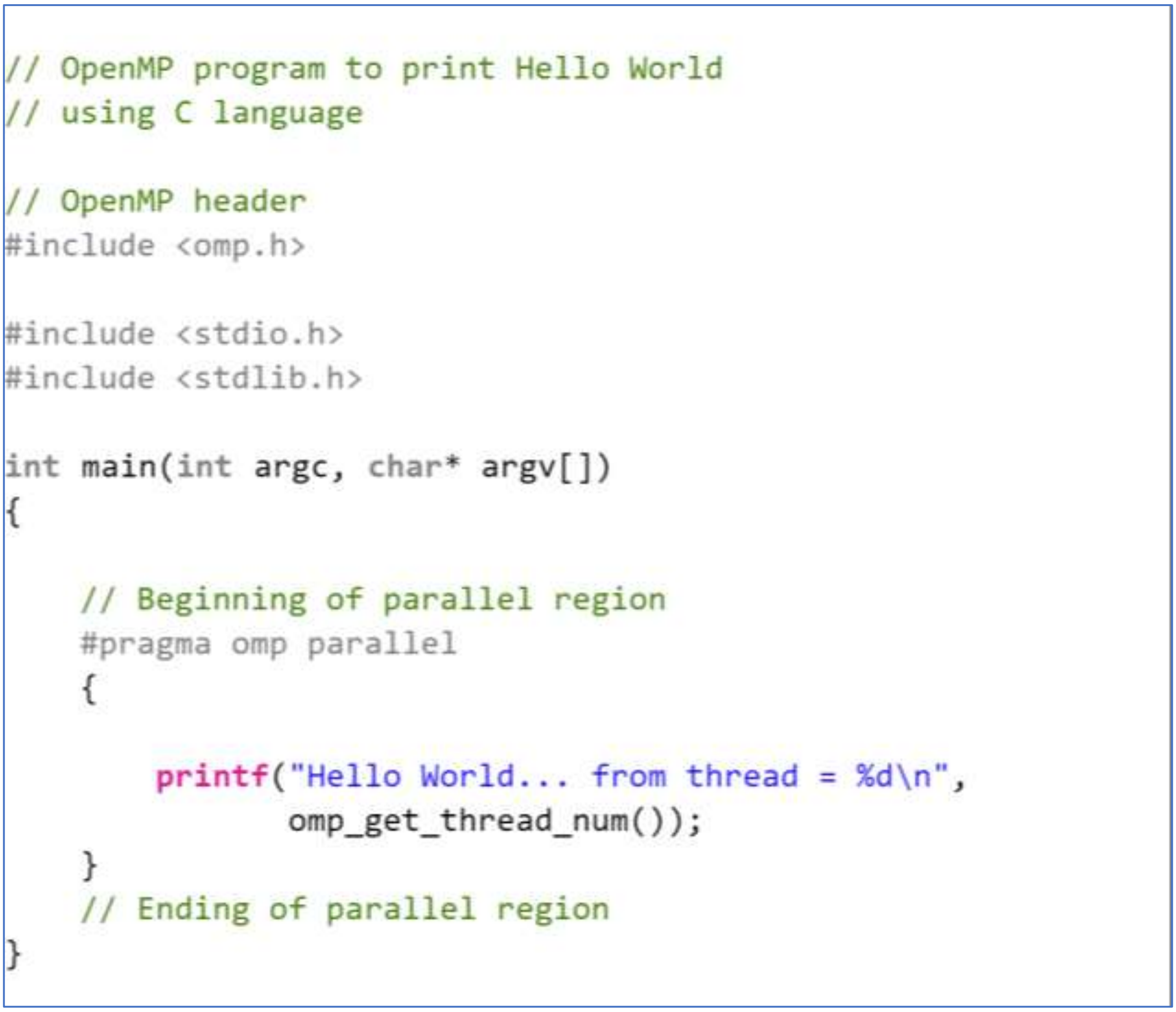
asma@asma-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
asma@asma-VirtualBox:~$ cd Desktop
asma@asma-VirtualBox:~/Desktop$ mpicc -o arraysum_c arraysum.c
asma@asma-VirtualBox:~/Desktop$ mpiexec -np 4 ./arraysum_c
Sum of array is : 55
asma@asma-VirtualBox:~/Desktop$ 
```

Name: Shivam Tiwari
Roll No: 5117060

Experiment No: 6

Aim: Using directives of MPI/OpenMP implement parallel programming for Hello World.



```
// OpenMP program to print Hello World
// using C language

// OpenMP header
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

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

    // Beginning of parallel region
    #pragma omp parallel
    {

        printf("Hello World... from thread = %d\n",
               omp_get_thread_num());
    }
    // Ending of parallel region
}
```

```
shivam@shivam -VirtualBox:~$ cd Desktop/
shivam@shivam -VirtualBox:~/Desktop
shivam@shivam -VirtualBox:~/Desktop$ export OMP_NUM_THREADS=5
shivam@shivam -VirtualBox:~/Desktop$ gcc -o hello -fopenmp
HelloworldOpenMP.c
shivam@shivam -VirtualBox:~/Desktop $ ./hello
Hello World... from thread = 2
Hello World... from thread = 4
Hello World... from thread = 1
Hello World... from thread = 0
Hello World... from thread = 3
```