**Lab-2**

**Distributed Systems**

**Suryanarayan.B**

**CB.EN.U4CSE19056**

1)

#include <mpi.h>

#include <stdio.h>

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

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

int world\_size;

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

// Get the rank of the process

int world\_rank;

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

// Get the name of the processor

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

// Print off a hello world message

printf("Hello world from processor %s, rank %d out of %d processors\n",

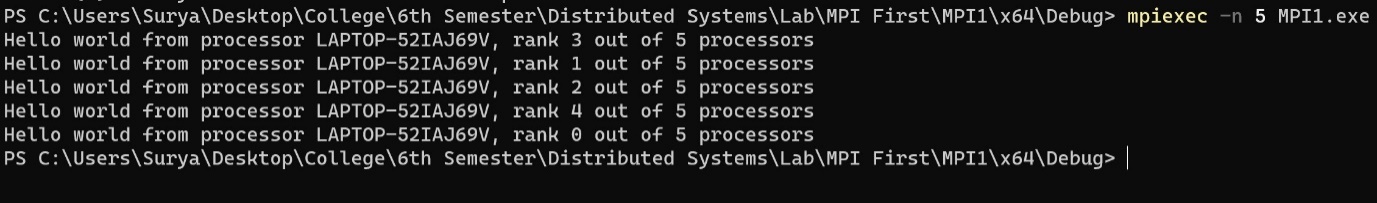
processor\_name, world\_rank, world\_size);

// Finalize the MPI environment.

MPI\_Finalize();

}

Output:



2)

#include <mpi.h>

#include <stdio.h>

int main() {

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the rank of the process

int world\_rank;

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

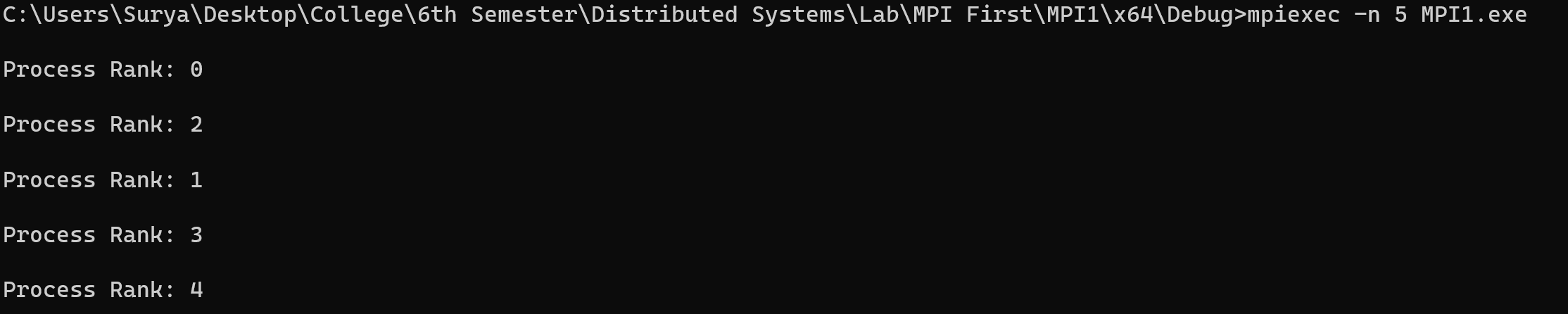
printf("\nProcess Rank: %d\n", world\_rank);

// Finalize the MPI environment.

MPI\_Finalize();

}

Output:



3)

#include <mpi.h>

#include <stdio.h>

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

int id, x,numpros,rootpros;

MPI\_Status status;

x= MPI\_Init(&argc, &argv);

x=MPI\_Comm\_size(MPI\_COMM\_WORLD, &numpros);

x=MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

if (id == 0)

{

printf("Process 1 is working !!!!!!!");

}

else

{

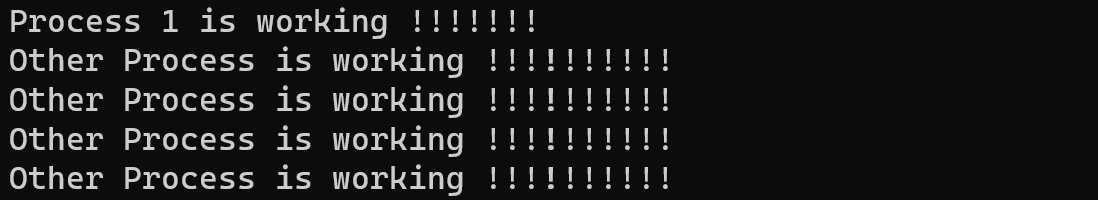
printf("Other Process is working !!!!!!!!!!");

}

x=MPI\_Finalize();

}

Output:



4)

#include <mpi.h>

#include <stdio.h>

#include<iostream>

using namespace std;

int main() {

int proc\_id, num\_processes, tag, src, dest, count;

int data;

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_processes);

// Get the rank of the process / process id

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_id);

// Set up communication params

tag = 100;

// Process 0 is sender, Process 1 is receiver

src = 0;

dest = 1;

count = 1;

if (proc\_id == src) {

data = 46920;

MPI\_Send(&data, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD);

cout<< "P\_ID" << proc\_id << " sent " << data <<"\n";

}

if (proc\_id == dest) {

MPI\_Recv(&data, count, MPI\_INT, src, tag, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

cout<< "P\_ID" << proc\_id << " received " << data << "\n";

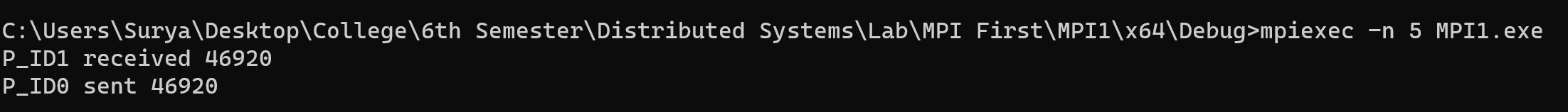
}

MPI\_Finalize();

return 0;

}

**Output:**

****

**5)**

#include <mpi.h>

#include <stdio.h>

#include<iostream>

using namespace std;

int main() {

int proc\_id, num\_processes, tag, src, dest, count;

int data, ack;

// Initialize the MPI environment

MPI\_Init(NULL, NULL);

// Get the number of processes

MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_processes);

// Get the rank of the process / process id

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &proc\_id);

// Set up communication params

tag = 100;

// Process 0 is sender, Process 1 is receiver

src = 0;

dest = 1;

count = 1;

if (proc\_id == src) {

data = 475638;

MPI\_Send(&data, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD);

cout << "Sender: P\_ID" << proc\_id << " sent " << data << "\n";

MPI\_Recv(&ack, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (ack == 1) {

cout << "Sender: Ack received from receiver : "<< ack << "\n";

}

}

if (proc\_id == dest) {

MPI\_Recv(&data, count, MPI\_INT, src, tag, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

cout << "Receiver: P\_ID" << proc\_id << " received " << data << "\n";

ack = 1;

MPI\_Send(&ack, count, MPI\_INT, src, tag, MPI\_COMM\_WORLD);

cout << "Receiver: P\_ID" << proc\_id << " sent " << ack << "\n";

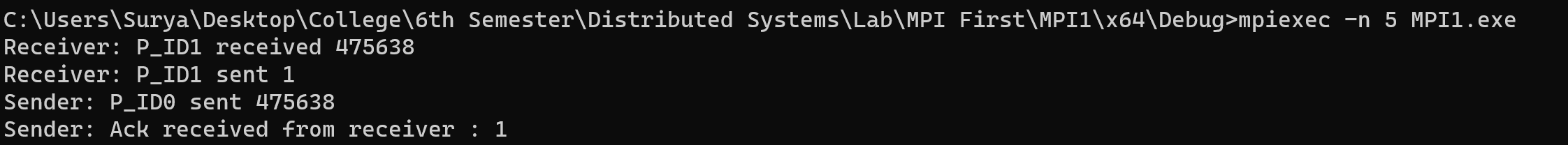
}

MPI\_Finalize();

return 0;

}

**Output:**

****

**6)**

#include <mpi.h>

#include <stdio.h>

#include<iostream>

using namespace std;

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

MPI\_Init(&argc, &argv);

int world\_size;

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

int world\_rank;

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

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

int name\_len;

MPI\_Get\_processor\_name(processor\_name, &name\_len);

if (world\_rank == 0) {

int ack = 0;

MPI\_Recv(

&ack,

1,

MPI\_INT,

1,

0,

MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE

);

cout << "deadlock in root";

}

else {

int ack = 0;

MPI\_Recv(

&ack,

1,

MPI\_INT,

0,

0,

MPI\_COMM\_WORLD,

MPI\_STATUS\_IGNORE

);

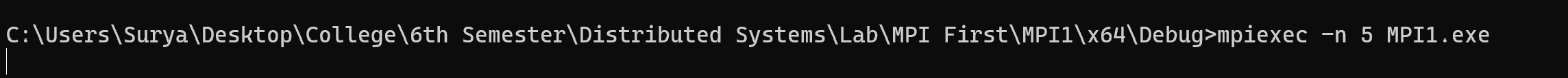
cout << "deadlock in child";

}

MPI\_Finalize();

}

**Output:**

****

**Deadlock**