**Class:** Final Year (Computer Science and Engineering)

**Year:** 2024-25 **Semester:** 1

**Course:** High Performance Computing Lab

## Practical No.8

PRN No :

Name :

# **Q1: Implement a MPI program to give an example of Deadlock.**

**Code :**

#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 received\_message;

if (world\_rank == 0) {

MPI\_Recv(&received\_message, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process 0 received message: %d\n", received\_message);

int message = 100;

MPI\_Send(&message, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

printf("Process 0 sent message: %d\n", message);

}

if (world\_rank == 1) {

MPI\_Recv(&received\_message, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process 1 received message: %d\n", received\_message);

int message = 200;

MPI\_Send(&message, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

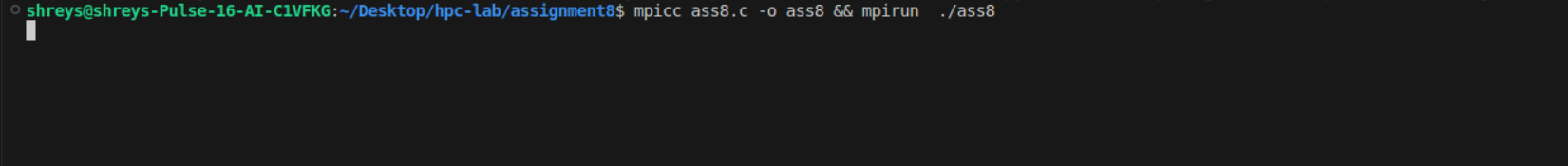
printf("Process 1 sent message: %d\n", message);

}

MPI\_Finalize();

return 0;

}

****

# **Q2. Implement blocking MPI send & receive to demonstrate Nearest neighbor exchange of data in a ring topology.**

# **CODE :**

#include <mpi.h>

#include <stdio.h>

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

MPI\_Init(&argc, &argv);

int world\_size;

int world\_rank;

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

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

int send\_data = world\_rank;

int recv\_data;

int next\_rank = (world\_rank + 1) % world\_size;

int prev\_rank = (world\_rank - 1 + world\_size) % world\_size;

MPI\_Send(&send\_data, 1, MPI\_INT, next\_rank, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&recv\_data, 1, MPI\_INT, prev\_rank, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

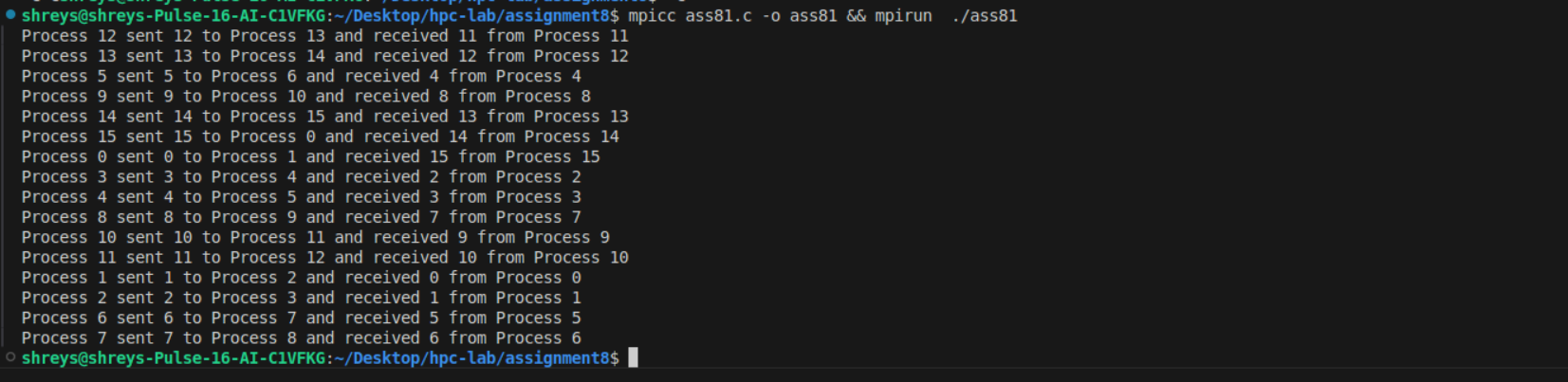
printf("Process %d sent %d to Process %d and received %d from Process %d\n",

world\_rank, send\_data, next\_rank, recv\_data, prev\_rank);

MPI\_Finalize();

return 0;

}



# **Q3. Write a MPI program to find the sum of all the elements of an array A of size**

**n. Elements of an array can be divided into two equals groups. The first [n/2]**

# **elements are added by the first process, P0, and last [n/2] elements the by second process, P1. The two sums then are added to get the final result.**

**CODE :**

**#include <mpi.h>**

#include <stdio.h>

#include <stdlib.h>

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

MPI\_Init(&argc, &argv);

int world\_size;

int world\_rank;

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

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

int n = 10;

int \*A = NULL;

if (world\_rank == 0) {

A = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i < n; i++) {

A[i] = i + 1;

}

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

int local\_n = n / 2;

int \*local\_A = (int\*)malloc(local\_n \* sizeof(int));

MPI\_Scatter(A, local\_n, MPI\_INT, local\_A, local\_n, MPI\_INT, 0, MPI\_COMM\_WORLD);

int local\_sum = 0;

for (int i = 0; i < local\_n; i++) {

local\_sum += local\_A[i];

}

int total\_sum = 0;

if (world\_rank == 0) {

int sum\_from\_P1;

MPI\_Recv(&sum\_from\_P1, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

total\_sum = local\_sum + sum\_from\_P1;

free(A);

} else if (world\_rank == 1) {

MPI\_Send(&local\_sum, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

}

if (world\_rank == 0) {

printf("Total sum of the array is: %d\n", total\_sum);

}

free(local\_A);

MPI\_Finalize();

return 0;

}

