Name: Shreeyash Shripad Dongarkar

PRN: 22510025

High Performance Computing Lab (B – 1)

Assignment – 6

Title: Installation of MPI & Implementation of Basic Functions of MPI

[GitHub Repository](https://github.com/TerminatorShri/22510025_HPCL)

Installation steps for MPI library and instructions regarding compilation and execution are written in README.md

Problem Statement 1:  
Implement a simple program to print ‘hello world’ by setting number of processors equal to 10.

1. #include <mpi.h>

2. #include <stdio.h>

3.

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

5.     MPI\_Init(&argc, &argv);

6.

7.     int world\_rank;

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

9.

10.     int world\_size;

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

12.

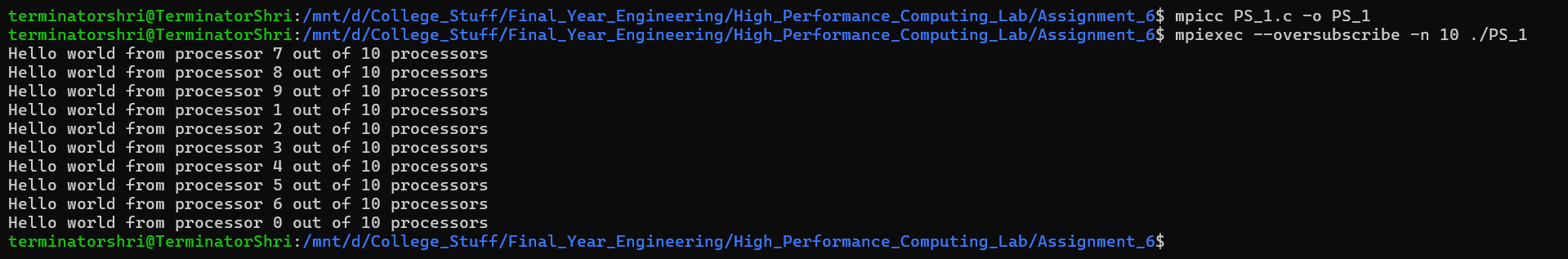
13.     printf("Hello world from processor %d out of %d processors\n", world\_rank, world\_size);

14.

15.     MPI\_Finalize();

16.     return 0;

17. }



Problem Statement 2:  
Implement a program to display rank and communicator group of five processes

1. #include <mpi.h>

2. #include <stdio.h>

3.

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

5.     int rank, size;

6.     MPI\_Group world\_group;

7.

8.     MPI\_Init(&argc, &argv);

9.

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

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

12.

13.     MPI\_Comm\_group(MPI\_COMM\_WORLD, &world\_group);

14.

15.     printf("Process %d of %d is in communicator MPI\_COMM\_WORLD\n", rank, size);

16.

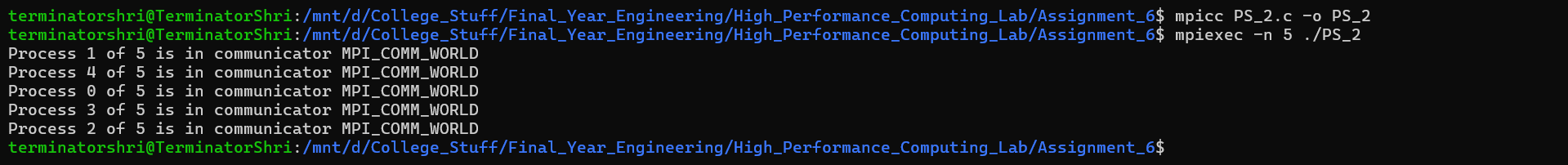
17.     MPI\_Group\_free(&world\_group);

18.     MPI\_Finalize();

19.

20.     return 0;

21. }



Problem Statement 3:  
Implement MPI Program to give an example of Deadlock.

1. #include <mpi.h>

2. #include <stdio.h>

3.

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

5.     int rank, size;

6.     int bigdata[1000000];

7.

8.     MPI\_Init(&argc, &argv);

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

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

11.

12.     if (size < 3) {

13.         if (rank == 0) printf("Run with 3 processes.\n");

14.         MPI\_Finalize();

15.         return 0;

16.     }

17.

18.     int next = (rank + 1) % 3;

19.     int prev = (rank + 2) % 3;

20.

21.     printf("Process %d sending to process %d\n", rank, next);

22.     MPI\_Send(bigdata, 1000000, MPI\_INT, next, 0, MPI\_COMM\_WORLD);

23.

24.     printf("Process %d receiving from process %d\n", rank, prev);

25.     MPI\_Recv(bigdata, 1000000, MPI\_INT, prev, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

26.

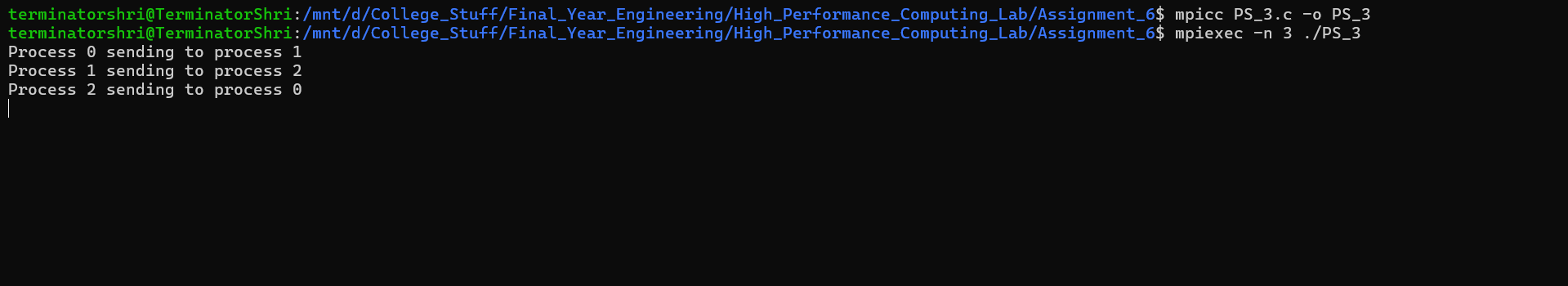
27.     printf("Process %d finished\n", rank);

28.

29.     MPI\_Finalize();

30.     return 0;

31. }



Problem Statement 4:  
Implement blocking MPI send & receive to demonstrate Nearest neighbour exchange of data in a ring topology.

1. #include <mpi.h>

2. #include <stdio.h>

3.

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

5.     int rank, size;

6.     int send\_data, recv\_data;

7.

8.     MPI\_Init(&argc, &argv);

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

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

11.

12.     if (size < 2) {

13.         if (rank == 0) printf("Run with at least 2 processes.\n");

14.         MPI\_Finalize();

15.         return 0;

16.     }

17.

18.     send\_data = rank;

19.     int next = (rank + 1) % size;

20.     int prev = (rank + size - 1) % size;

21.

22.     if (rank % 2 == 0) {

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

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

25.     } else {

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

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

28.     }

29.

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

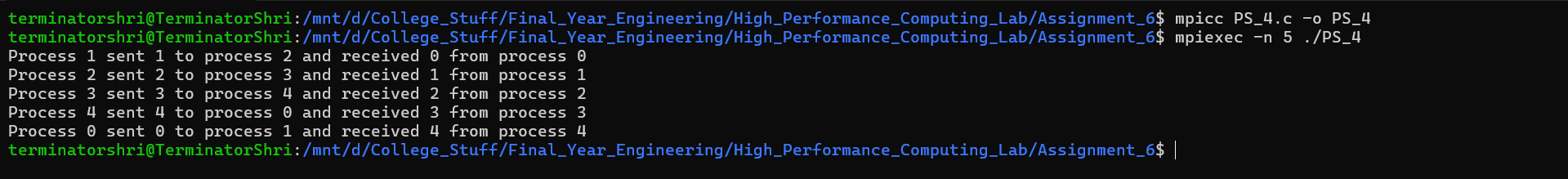
31.            rank, send\_data, next, recv\_data, prev);

32.

33.     MPI\_Finalize();

34.     return 0;

35. }



Problem Statement 5:  
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.

1. #include <mpi.h>

2. #include <stdio.h>

3. #include <stdlib.h>

4.

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

6.     int rank, size;

7.     MPI\_Init(&argc, &argv);

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

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

10.

11.     if (size != 2) {

12.         if (rank == 0) printf("This program requires exactly 2 processes.\n");

13.         MPI\_Finalize();

14.         return 0;

15.     }

16.

17.     int n = 10;

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

19.     int local\_sum = 0;

20.

21.     if (rank == 0) {

22.         for (int i = 0; i < n/2; i++)

23.             local\_sum += A[i];

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

25.     } else if (rank == 1) {

26.         for (int i = n/2; i < n; i++)

27.             local\_sum += A[i];

28.         int sum0;

29.         MPI\_Recv(&sum0, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

30.         int total\_sum = local\_sum + sum0;

31.         printf("Sum of array elements = %d\n", total\_sum);

32.     }

33.

34.     MPI\_Finalize();

35.     return 0;

36. }

