Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical
No.8

PRN No : 21510042

Name : Omkar Rajesh Auti

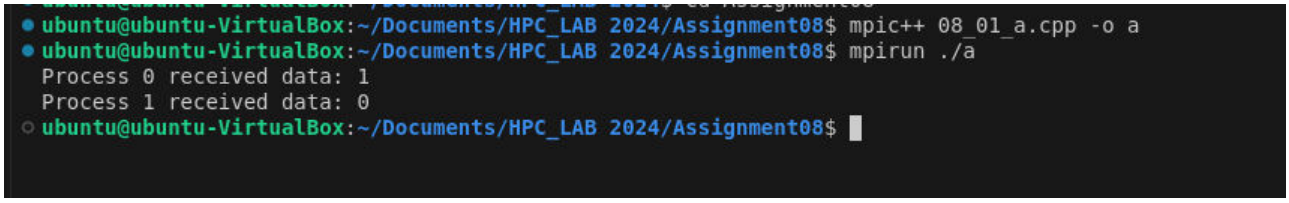
Q1: Implement a MPI program to give an example of Deadlock.

Code:

```
#include <mpi.h>
#include <iostream>

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

int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);

if (size != 2) {
std::cerr << "This program requires exactly 2 processes.\n";
MPI_Abort(MPI_COMM_WORLD, 1);
}

int send_data = rank;
int recv_data;

if (rank == 0) {
// Process 0 sends to Process 1 and waits for Process 1 to send
MPI_Send(&send_data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
MPI_Recv(&recv_data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
} else if (rank == 1) {
// Process 1 sends to Process 0 and waits for Process 0 to send
MPI_Send(&send_data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
MPI_Recv(&recv_data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

std::cout << "Process " << rank << " received data: " << recv_data <<
std::endl;
```

```
MPI_Finalize();
return 0;
}
```

Screenshot:

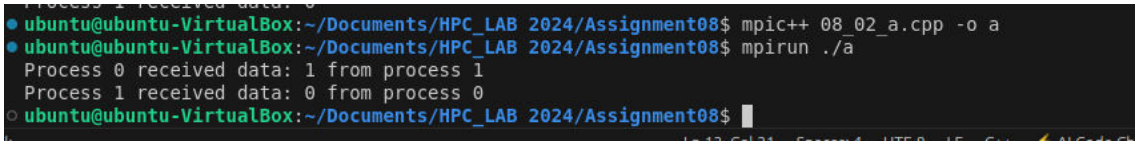


Q2. Implement blocking MPI send & receive to demonstrate Nearest neighbor exchange of data in a ring topology.

Code:

```
#include <mpi.h>
#include <iostream>

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);

int send_data = rank;
int recv_data;
int next = (rank + 1) % size;
int prev = (rank - 1 + size) % size;

// Exchange data with neighbors in a blocking fashion
MPI_Send(&send_data, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
MPI_Recv(&recv_data, 1, MPI_INT, prev, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

std::cout << "Process " << rank << " received data: " << recv_data << " from
process " << prev << std::endl;

MPI_Finalize();
return 0;
}
```

Screenshot:



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 <iostream>

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);

const int n = 10;
int A[n];
int local_sum = 0;

// Initialize the array in process 0
if (rank == 0) {
for (int i = 0; i < n; ++i) {
A[i] = i + 1; // Example: A = {1, 2, 3, ..., 10}
}
}

// Broadcast the array to both processes
MPI_Bcast(A, n, MPI_INT, 0, MPI_COMM_WORLD);

// Divide the work between the two processes
int start = (rank == 0) ? 0 : n / 2;
int end = (rank == 0) ? n / 2 : n;

// Each process calculates its local sum
for (int i = start; i < end; ++i) {
local_sum += A[i];
}

std::cout << "Process " << rank << " local sum: " << local_sum << std::endl;
```

```
// Reduce the local sums to get the global sum
int global_sum = 0;
MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

if (rank == 0) {
std::cout << "Global sum: " << global_sum << std::endl;
}

MPI_Finalize();
return 0;
}
```

Screenshot: