Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical No.8

PRN No : 2020BTECS00010
Name : Abhijeet kamalekar

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

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

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

int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

if (world_rank == 0) {
// Process 0 tries to send a message to Process 1
int msg = 0;
MPI_Send(&msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
printf("Process 0 sent a message.\n");
// Now waits to receive from Process 1 (deadlock occurs)
MPI_Recv(&msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
} else if (world_rank == 1) {
// Process 1 tries to send a message to Process 0
int msg = 1;
MPI_Send(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
printf("Process 1 sent a message.\n");
// Now waits to receive from Process 0 (deadlock occurs)
MPI_Recv(&msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
MPI_Finalize();
return 0;
}
```

Output:

```
abhijeet@abhijeet-VirtualBox:~/MPI$ mpirun -np 2 ./8_c
Process 1 sent a message.
Process 0 sent a message.
abhijeet@abhijeet-VirtualBox:~/MPI$
```

Q2. Implement blocking MPI send & receive to

demonstrate Nearest neighbor exchange of data in a ring topology.

```c
#include <stdio.h>
#include <mpi.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 send_data = world_rank; // Data to send
int recv_data;

// Send to the next neighbor
int next_rank = (world_rank + 1) % world_size; // Nearest neighbor
MPI_Send(&send_data, 1, MPI_INT, next_rank, 0, MPI_COMM_WORLD);
// Receive from the previous neighbor
int prev_rank = (world_rank - 1 + world_size) % world_size; // Nearest neighbor
MPI_Recv(&recv_data, 1, MPI_INT, prev_rank, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

// Print received data
printf("Process %d received data %d from Process %d\n", world_rank, recv_data,
prev_rank);

MPI_Finalize();
return 0;
}
```

output:

```
abhijeet@abhijeet-VirtualBox:~/MPI$ mpirun -np 4 ./8_2
Process 0 received data 3 from Process 3
Process 3 received data 2 from Process 2
Process 1 received data 0 from Process 0
Process 2 received data 1 from Process 1
```

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.

```
#include <stdio.h>
#include <mpi.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);

// Size of the array (must be even for this example)
const int n = 8; // Example size
int A[n];
int local_sum = 0;

// Initialize the array only in process 0
if (world_rank == 0) {
// Fill the array with some values
for (int i = 0; i < n; i++) {
A[i] = i + 1; // Array will be {1, 2, 3, 4, 5, 6, 7, 8}
}
}

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

// Each process calculates its local sum
if (world_rank == 0) {
// Process 0 sums the first half
for (int i = 0; i < n / 2; i++) {
local_sum += A[i];
}
} else if (world_rank == 1) {
// Process 1 sums the second half
for (int i = n / 2; i < n; i++) {
local_sum += A[i];
}
}

// Reduce the local sums to the total sum in process 0
int total_sum;
MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

```
// Print the total sum in process 0
if (world_rank == 0) {
printf("Total sum of array elements: %d\n", total_sum);
}

MPI_Finalize();
return 0;
}
```

output:

```
abhijeet@abhijeet-VirtualBox:~/MPI$ mpirun -np 2 ./8_3
Total sum of array elements: 36
abhijeet@abhijeet-VirtualBox:~/MPI$
```