Class: Final Year (Computer Science and Engineering)

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

Course: High Performance Computing Lab

Practical No.8

PRN No : 21510111

Name : Siddharth Salunkhe

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

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

int main(int argc, char** argv) {
    int rank, size;
    int send_data[100000];  // Large array to force blocking
    int recv_data[100000];

    // Initialize MPI environment
    MPI_Init(&argc, &argv);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Ensure we have at least 2 processes
    if (size < 2) {
        printf("This example requires at least two processes.\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    if (rank == 0) {
        // Process 0 tries to send to Process 1, then receive from Process 1
        MPI_Send(send_data, 100000, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(recv_data, 100000, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    } else if (rank == 1) {
        // Process 1 tries to send to Process 0, then receive from Process 0
        MPI_Send(send_data, 100000, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Recv(recv_data, 100000, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    // Output the results (this won't be reached due to deadlock)
    printf("Process %d completed communication.\n", rank);

    // Finalize MPI environment
    MPI_Finalize();

    return 0;
}
```

Output:

```
*[main][~/acad/hpc-lab/as8]$ mpirun -np 2 ./1
^C%
*[main][~/acad/hpc-lab/as8]$
```

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

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

int main(int argc, char** argv) {
    int rank, size;
    int send_data, recv_data;
    int left, right;

    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Get the number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Define the left and right neighbors in the ring
    left = (rank - 1 + size) % size;   // Left neighbor
    right = (rank + 1) % size;         // Right neighbor

    // Data to send (rank of the current process)
    send_data = rank;

    // Perform nearest neighbor exchange using blocking communication
    MPI_Send(&send_data, 1, MPI_INT, right, 0, MPI_COMM_WORLD);  // Send to the right neighbor
    MPI_Recv(&recv_data, 1, MPI_INT, left, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  // Receive

    // Output the result
    printf("Process %d received data %d from process %d\n", rank, recv_data, left);

    // Finalize the MPI environment
    MPI_Finalize();

    return 0;
}
```

Output :

```
*[main][~/acad/hpc-lab/as8]$ mpirun --oversubscribe -np 5 ./2
Process 4 received data 3 from process 3
Process 3 received data 2 from process 2
Process 0 received data 4 from process 4
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.

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

int main(int argc, char** argv) {
    int rank, size, n = 10;
    int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  // Example array of size 10
    int local_sum = 0, global_sum = 0;
    int i, start, end;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size != 2) {
        if (rank == 0) {
            printf("This program requires exactly 2 processes.\n");
        }
        MPI_Finalize();
        return -1;
    }

    if (rank == 0) {
        start = 0;
        end = n / 2;
    } else if (rank == 1) {
        start = n / 2;
        end = n;
    }

    for (i = start; i < end; i++) {
        local_sum += A[i];
    }
    printf("Process %d local sum: %d\n", rank, local_sum);

    if (rank == 0) {
        int received_sum;
        MPI_Recv(&received_sum, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        global_sum = local_sum + received_sum;
        printf("Final sum: %d\n", global_sum);  // P0 prints the final result
    } else if (rank == 1) {
        MPI_Send(&local_sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();

    return 0;
}
```

Output :

```
*[main][~/acad/hpc-lab/as8]$ mpirun -np 2 ./3
Process 0 local sum: 15
Process 1 local sum: 40
Final sum: 55
```